NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

Case Study on Rapid Software Prototyping and Automated Software Generation: An Inertial Navigation System

by

Herbert Günterberg

June 1989

Thesis Advisor: Luqi

Approved for public release; distribution is unlimited.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION Naval Postgraduate School; 6b OFFICE SYMBOL 52	7a NAME OF MONITORING ORGANIZATION [1] National Science Foundation [2] Office of Naval Research [3] NPS Research Council
6c ADDRESS (City, State, and ZIP Code) Monterey, CA 93943	7b ADDRESS (City, State, and ZIP Code) [1] Washington, D.C. 20550 [2] 800 Quincy Street, Arlington, VA 22217-5000 [3] Monterey, CA 93943
8a NAME OF FUNDING/SPONSORING ORGANIZATION; 8b OFFICE SYMBOL (If applicable)	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code)	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)
Case Study on Rapid Software Prototyping and Automated Software Generation: An Inertial Navigation System

12 PERSONAL AUTHOR(S) Günterberg, Herbert

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Masters Thesis	FROM ______ TO ______	1989 June	98

16 SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	(Rapid Prototyping, Specification Language, Graphic Editor, Translator, Execution Support System, Real-Time Programming, Parallell Execution, Ada)

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

The discipline of software engineering is on the move from an "art" to an engineering science based on mathematical rules. Along this way methods of rapid prototyping and tools for automatic program generation are being developed to aid the process of software development. This thesis takes a real life example of an Inertial Navigation System and develops it according to the automation principles for computer aided software development. The techniques of rapid software prototyping are also applied to the same problem. The software prototype of the Inertial Navigation System can further be run through The Computer Aided Prototyping System (CAPS) to mechanically generate Ada software. All implementation work is done in Ada as required by DoD for all embedded weapon systems. The two approaches will be integrated for analysis.

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT ☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	21 ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED	
22a NAME OF RESPONSIBLE INDIVIDUAL Luqi	22b TELEPHONE (Include Area Code) (408)646-2735	22c OFFICE SYMBOL 52LQ

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

Office 1986—606-24.

Approved for public release; distribution is unlimited

Case Study on Rapid Software Prototyping and Automated Software Generation: An Inertial Navigation System

by

Herbert Günterberg
Lieutenant Commander, Federal German Navy

Submitted in partial fullfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1989

Author: ______________________
Herbert Günterberg

Approved by: ______________________
Luqi, Thesis Advisor

____
Valdis Berzins, Second Reader

____
Robert B. McGhee, Chairman, Department of Computer Science

____
Kneale T. Marshall, Dean of Information and Policy Sciences

ABSTRACT

The discipline of software engineering is on the move from an "art" to an engineering science based on mathematical rules. Along this way methods of rapid prototyping and tools for automatic program generation are being developed to aid the process of software development. This thesis takes a real life example of an Inertial Navigation System and develops it according to the automation principles for computer aided software development. The techniques of rapid software prototyping are also applied to the same problem. The software prototype of the Inertial Navigation System can further be run through The Computer Aided Prototyping System (CAPS) to mechanically generate Ada software. All implementation work is done in Ada as required by DoD for all embedded weapon systems. The two approaches will be integrated for analysis.

TABLE OF CONTENTS

LIST OF FIGURES

THESIS DISCLAIMER

- Ada is a registered trademark of the United States Government Ada Joint Program Office.
- INTEGRADA is a trademark of AETECH, Inc.
- Meridian AdaVantage is a trademark of Meridian Software Systems, Inc.
- Sun Workstation is a registered trademark of Sun Microsystems Inc.

The source code developed in this thesis is in not meant for operational use, but for an academic purpose, therefore anybody who is going to use the code or part of it shall be advised to check the correctness for the particular application. The author does not accept any responsibility beyond the academic environment.

ACKNOWLEDGEMENT

... to my wife Gudrun and my sons Andreas and Daniel for reminding me sometimes, that there is a world besides computers.

I. INTRODUCTION

A. THE SOFTWARE CRISIS

What is the software crisis? To explain this a look at the development of computers will be helpful. The early machines had very little memory capacity, therefore the programs which could run on these machines had to be restricted in their need for memory as well (the technique of overlays had not evolved then). Since programs were small it was very easy for a single person to comprehend a program in its entirety. In those days programming was more of an art than a science. The major portion of the cost of a computer system was associated with hardware. Computers have come a long way since then. Memory capacity has increased to a level that was considered impossible only a few years ago. Presently hardware technology advances at a speed of improving the memory capacity and speed by a factor of two about every two years.

Unfortunately the software side of computer systems has not been able to keep up with hardware development. More and more problems are considered to be suitable for automation and computer application, the problem domain expanded. Soon no one person was able to comprehend a software system as a single person, but the techniques used were the same as in the beginning. This led to the software crisis, the symptoms are described by Booch [Ref. 1:p. 8] as:

- *Responsiveness.* Computer-based systems often do not meet user needs.
- *Reliability.* Software often fails.
- *Cost.* Software costs are seldom predictable and are often perceived as excessive.
- *Modifiability.* Software maintenance is complex, costly, and error prone.
- *Timeliness.* Software is often late and frequently delivered with less-than-promised capability.

- *Transportability*. Software from one system is seldom used in an other, even when similar functions are required.
- *Efficiency*. Software development efforts do not make optimal use of the resources involved (processing time and memory space).

Having stated the symptoms of the crisis, the next question must be about the causes, which are summarized by Devlin [Ref. 2:p. 2] as:

- Failure of organizations to understand the life-cycle implications of software development.
- A shortage of personnel trained in software engineering.
- The von Neumann architectures of most of our machines discourage the use of modern programming practices.
- The tendency of organizations to become entrenched in the use of archaic programming languages and practices.

This research explores two efforts which have been undertaken over the last years to solve the above stated problems. The following two sections will give a brief overview.

B. RAPID PROTOTYPING

One effort to increase software development productivity is rapid software prototyping. It is especially worthwhile in the development of hard real time systems. In traditional Software production, a system has to be fully implemented to confirm that the final product meets the requirements. The idea behind rapid prototyping is to create a prototype of the proposed system to verify that the real time behavior demanded by the customer is feasible under the imposed constraints. This can save tremendous amounts of resources in terms of money and work, because the feasibility of the system is verified **before** the actual design and implementation of the system is undertaken. Design errors are magnitudes cheaper to correct at this level compared to redesigning and recoding of a finished product which doesn't meet the customer requirements.

One such system for rapid prototyping. called CAPS (**C**omputer **A**ided **P**rototyping **S**ystem) which is based on PSDL (**P**rototype **S**ystem **D**escription **L**anguage) is presently under development in a research project at NPS. Background information on the CAPS and a more in depth reference to PSDL can be found in [Ref. 3, 4, 5, 6, 7, 8]. In this thesis features of PSDL and CAPS concepts will be explained only the extend that is necessary for understanding and these explanations will be given as the need arises.

C. FORMAL SOFTWARE ENGINEERING

Another approach, which consideres the complete software lifecycle anmd not just the prototyping aspect of software development was developed by Berzins and is described in [Ref. 9]. The following is a short extract to summarize the key concepts of his approach.

DEFINITION:

Software Engineering is the application of science and mathematics to the problem of making computers useful to people by means of software.[Ref. 9:p. 1-1]

Software development can be viewed as a five stage process. The concept and the relations between the different stages can be seen in [Figure 1:p. 3]

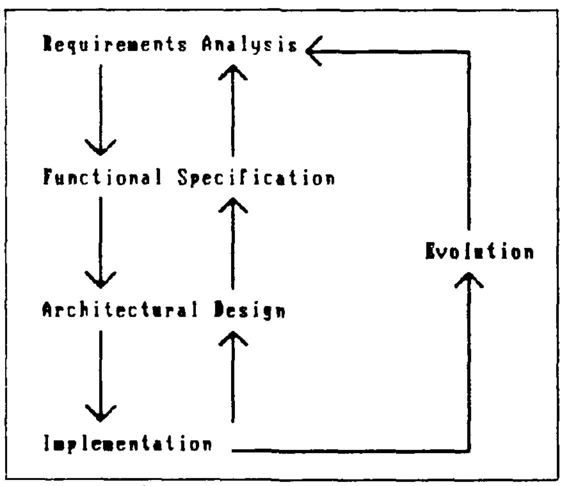

Figure 1: Software Development Process

The downarrows show the normal flow of execution, the uparrows represent details gained at a later stage, which require the repetition of an earlier step. The long arrow labeled " Evolution" demonstrates that every software product is subject to change due to altered operating conditions or user needs.

Each of these five steps is associated with certain goals, which are described in [Ref 9:p. 12] as:

- *REQUIREMENTS ANALYSIS*: Is the process of determining and documenting the user needs and constraints.
- *FUNCTIONAL SPECIFICATION*: Is the process of proposing and formalizing a proposed system interface for meeting the customer needs.
- *ARCHITECTURAL DESIGN*: Is the process of decomposing the system into modules and defining internal interfaces.
- *IMPLEMENTATION*: Is the process of producing a program for each module.
- *EVOLUTION*: Is the process of adapting the system to the changing needs of the customer.

II. THE PROTOTYPE APPROACH

A. ABOUT CAPS

CAPS can be characterized as a composition of separate tools which provide the means to create a prototype of a software system in a fraction of the time the actual development would take. It is not meant to replace a good software development environment, but to aid it and make it even better. The prototyping system as mentioned in Chapter I, has not yet been completely implemented; therefore a summary of the capabilities of the completed system will be given. A description of the currently cperational parts that were used for this thesis as well as the development state of the other parts will follow. The system incorporates these tools:

- User Interface
- Graphic Editor
- Syntax Directed Editor
- Language Translator
- Debugger
- Static Scheduler
- Dynamic Scheduler
- Software Base Management System
- Design Database

The user interface ties all the tools together. It takes care of the proper filename conventions and file formats to be passed between the tools. For the development of a new prototype, the designer would start with the graphic editor tool.

The graphic editor supports a graphical representation of the dataflow model underlying the PSDL language. Building blocks of the graphic language are nodes and arcs. Nodes represent functions or state machines, collectively called operators. Arcs represent dataflows among others, external inputs or outputs.

Once in the graphic editor, the mouse becomes the primary input device for control over the editor, whereas text input is entered via the keyboard into designated windows. The following operations are available to the user:

- for file management:
  - LOAD EXISTING - to retrieve a previously created file for modification.
  - STORE - to store the current graphical representation of a prototype.
  - QUIT - to return to the user interface.
- for editing:
  - DRAW OPERATOR - to draw an operator. Each operator must have a unique identifier and a time constraint which is the maximum execution time associated with the operator.
  - DRAW DATA FLOW - to draw a data flow between two operators, it also must have a unique identifier and, since the direction of a data flow is important, it must be taken care of during the drawing process. The data flow has to start at its originating operator and end at its destination operator.
  - DRAW SELF LOOP - to draw a self loop, which is the graphic representation of a state variable, a PSDL construct necessary to describe a state machine.
  - DRAW INPUT - to draw an external input into the system. This is also a data flow with the difference that it doesn't flow from one operator to another, but from an external source e.g. user, other software or hardware system.
  - DRAW OUTPUT - to draw an external output, similar to drawing an input. except for the direction.

A system screen dump taken during the creation of operator INS is shown in [Figure 2:p. 7]. After leaving the graphic editor certain files are created, whose contents will be described during the actual development process later on.

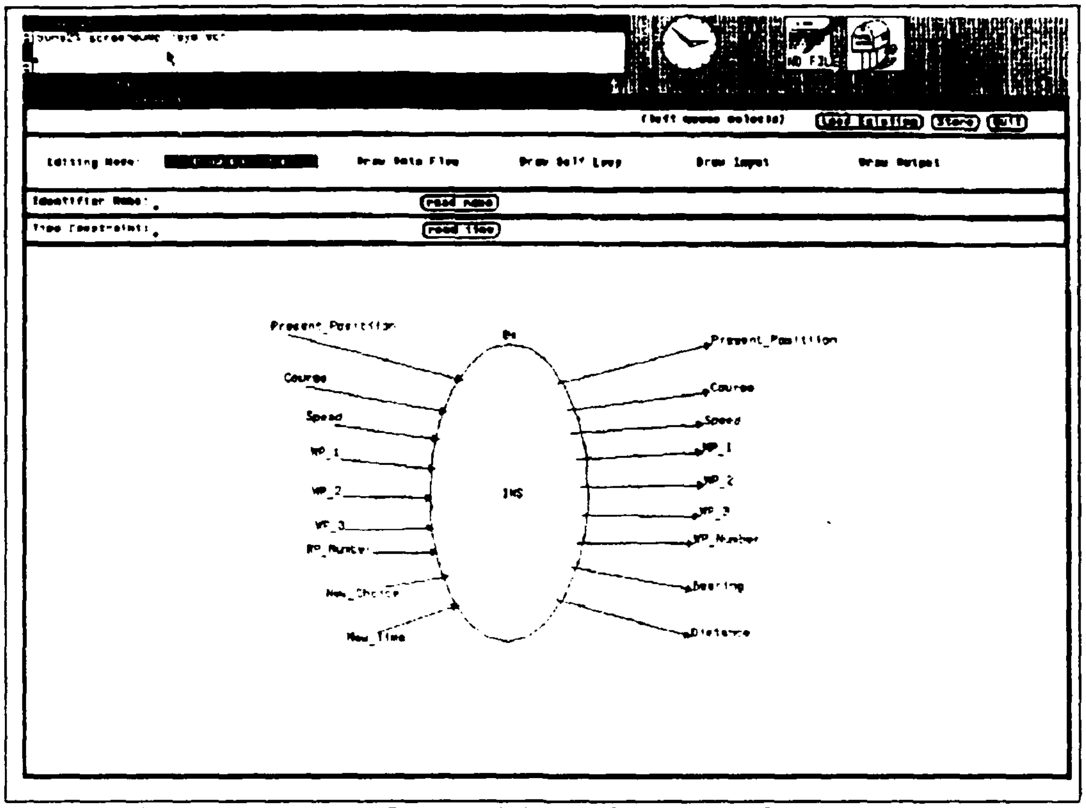

Figure 2: Screendump from Graphic Editor - Operator INS

Output from the graphic editor in textual form is feed into the syntax directed editor, whose main purpose is to guarantee the completion of a syntactically correct PSDL program. It assists the user in adding information into the prototype which is not easily representable in graphic form e.g. periodical behavior of an operator, type declarations for data flows of all three kinds and triggering conditions. The importance of syntactically correct PSDL programs becomes obvious in the employment of the next tool, the language translator, which relies on this property to translate a PSDL program into executable Ada code.

The static scheduler takes the output from the language translator and creates a time schedule for the execution of all time-critical operators and organizes it so that all timing constraints will be met during execution if possible. All non time-critical operators are handled by the dynamic scheduler. It checks the static schedule for any unused time slots and schedules non time-critical operators for execution during those times. The execution of non time-critical operators may be suspended before completion, when the static scheduler needs the resources for a time-critical operator.

Whenever there is a conflict during the creation of the schedules or the execution of the prototype, the debugger is invoked, to give the user a chance to solve the conflict dynamically on line, instead of breaking off execution and thereby forcing the repetition of the whole scheduling process from the beginning.

Two databases complete the system. The software base contains reusable Ada components, which are searched for using the PSDL specification of an operator, the design data base keeps track of the prototype currently under construction, it maintains this information by storing PSDL specifications.

The user interface and graphic editor are completely implemented and were used for this thesis. The language translator is implemented as well, but does not yet include all the constructs used in this project such as composite data types, therefore it was not used. For all the other components designs exist, some are partially implemented, but not operational.

B. THE INS PROTOTYPE DEVELOPMENT IN PSDL

The first tool to be used in the prototype development is the graphic editor. It is implemented on a SUN workstation and makes extensive use of its windowing and graphics capabilities. The editor is invoked from the main menu of the user interface with option "construct" [Ref. 3]. This in turn invokes the 'GE' script. At the top level design of INS only one operator exists with all inputs and outputs intended for the complete system. No timing constraints were placed on operator INS. The inputs and outputs are data streams of type data flow. Streams behave like FIFO queues (first-in-first-out) with a fixed length of one element. thereby implying. that a new value can only be added to the queue. after the old value has been read. For further explanations see [Ref. 6:p. 9]. After all the entities have been entered into the graphic editor, the picture is saved in the file SYS.G. The GE script partially produces the syntactically correct PSDL specification for operator INS, where only the data types for the input and output data have to be specified, which would normally be done in the syntax directed editor. Since it is not operational at this time, the editing has to be done manually in a standard word processor.

The following represents the specification, which is partially created by GE and completed manually by adding the datatypes:

```
OPERATOR INS

SPECIFICATION

   INPUT    Present_Position    : POSITION;
            Course              : FLOAT;
            Speed               : INTEGER;
            WP_1                : POSITION;
            WP_2                : POSITION;
            WP_3                : POSITION;
            WP_number           : INTEGER;
            New_time            : TIME;
            New_choice          : INTEGER;

   OUTPUT   Present_Position    : POSITION;
            Course              : FLOAT;
            Speed               : INTEGER;
            WP_1                : POSITION;
            WP_2                : POSITION;
            WP_3                : POSITION;
            WP_number           : INTEGER;
            Bearing             : FLOAT;
            Distance            : FLOAT;
END
```

Since the design database does not contain an implementation for operator INS, it needs to be decomposed. The graphic representation is provided in [Figure 3:p. 10]. New constructs used in the decomposition are state variables, which are represented

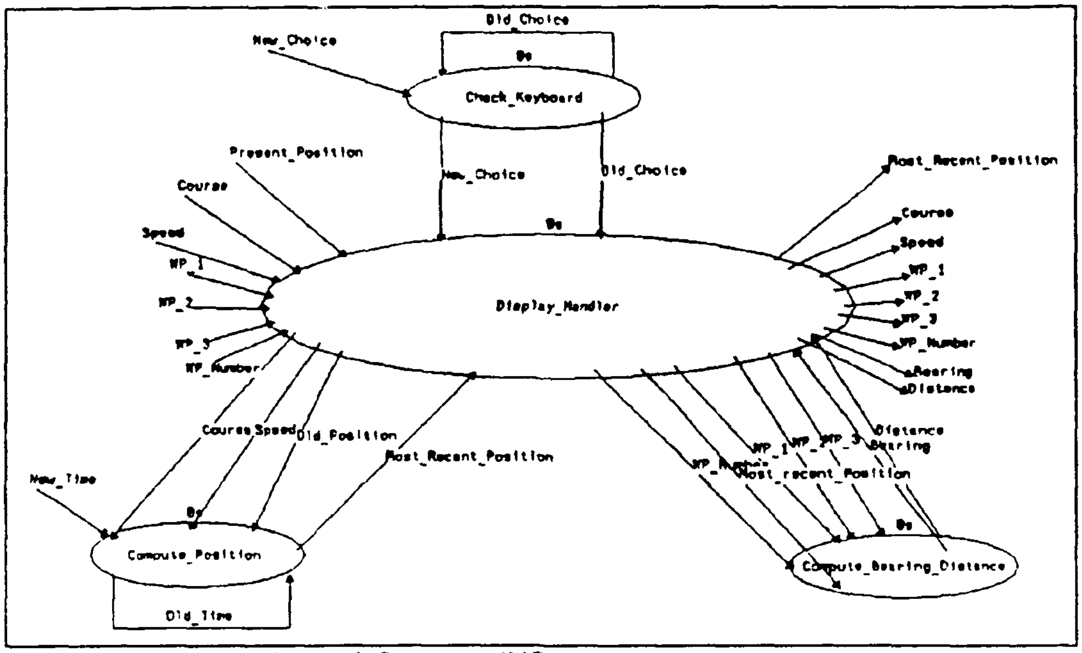

Figure 3: Decomposition of Operator INS

as self loops and data streams between operators, but unlike the data streams into and out of operator INS these are sampled data streams, which means that the data are buffered. A new value can be written to the buffer regardless of whether the old value has been read or not. The buffer can be read as often as needed, always providing the most recent data value.

As soon as a value has been placed into it for the first time, a read operation does not destroy the old data value, whereas a write operation will replace the old value with the most recent one. For further explanations see [Ref.6:p. 9]. In this example operators CHECK_KEYBOARD, COMPUTE_POSITION AND COMPUTE_BEARING_DISTANCE are atomic and need no further decomposition. Atomic operators are those which are already in the design database or can be easily implemented in Ada.

CHECK_KEYBOARD as its name suggests checks the keyboard for an interrupt, which in turn directs the flow of control for the lower levels of the system depending on user input. If no new interrupt is sensed, the control is directed according to the last interrupt. This scheme turns the system in its entirety into a state machine. Control of the lower levels is executed via the data streams OLD_CHOICE or NEW_CHOICE.

COMPUTE_POSITION is an independent process which updates the present position of the aircraft using the velocity values received by the system, the last valid present position, called OLD_POSITION, or a new position entered by the user. It produces a new present position, called MOST_RECENT_POSITION. The reason for using three different names for the same entity, a present position, lies in the naming conventions used in the graphic editor and PSDL itself. If the same name is used for several data streams (overloading) the system treats all those streams as being the same which is not really the case.

COMPUTE_BEARING_DISTANCE is another independent process working on the MOST_RECENT_POSITION, a WP_NUMBER which represents a user choice and the respective waypoint data contained in WP_1, WP_2 or WP_3 respectively. The outputs BEARING and DISTANCE are stored in their appropriate buffers.

Operator DISPLAY_HANDLER is composite. [Figure 3:p 11] gives the graphic representation. All operators at this level are atomic; they comprise input and output for the system. The left column contains the operators responsible for input. In the middle column the data buffers are grouped together. Inputs to these buffers are all of type sampled data stream. Operators for system output are in the right column.

A word of explanation about the data buffers used is in order here. The fact that the above mentioned data streams are considered to be sampled data stream implies that they are inherently buffered, therefore no buffers as depicted in [Figure 3:p. 10] need to be explicitly mentioned, however they are included here for a better understanding of the system layout. In the strict sense of PSDL the middle row in the figure could be eliminated without changing the meaning or behavior of the overall system.

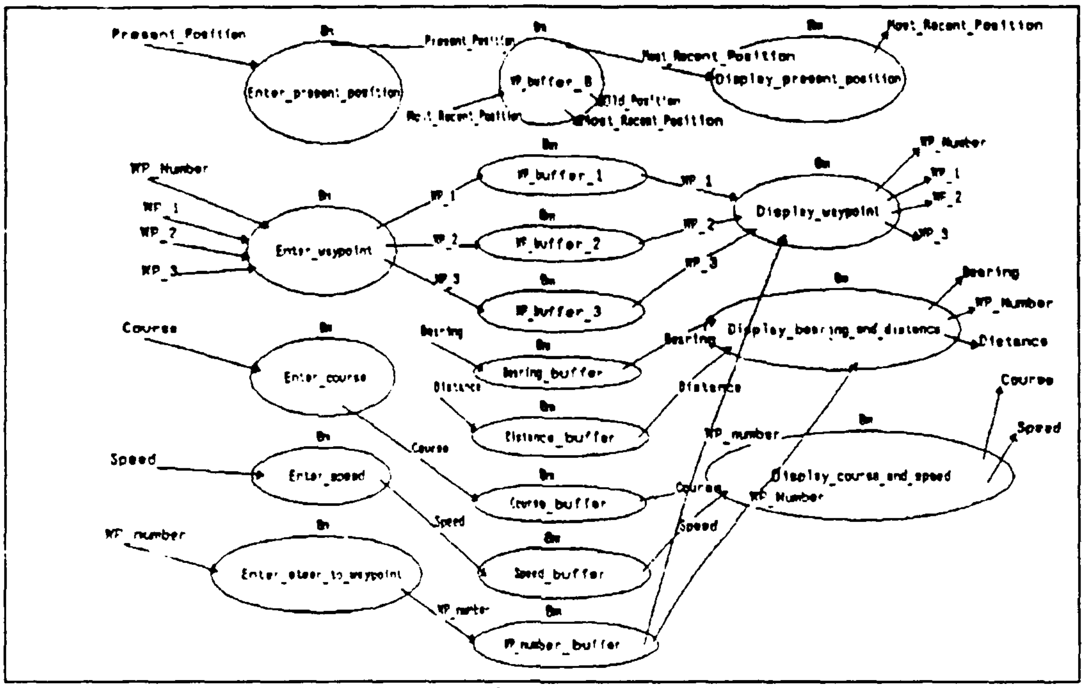

Figure 4: Internal Representation of Operator DISPLAY_HANDLER

In addition to creating the PSDL specification the file PSDL.LINKS is created, which contains the textual representation of the operators in the form of link statements connecting the different operators. At the end is a list of all internal data streams. its contents for the decomposition of OPERATOR INS is shown below and on the next page.

OPERATOP INS:

Old_choice.Check_keyboard --> Check_keyboard
Old_choice.Check_keyboard --> Display_handler
New_choice.Check_keyboard --> Display_handler
Bearing.Compute_bearing_distance --> Display_handler
Distance.Compute_bearing_distance --> Display_handler
Speed.Display_handler --> EXTERNAL
Speed.Display_handler --> Compute_position
Course.Display_handler --> EXTERNAL
Course.Display_handler --> Compute_position
Old_Position.Display_handler --> Compute_position
Bearing.Display_handler --> EXTERNAL
Distance.Display_handler --> EXTERNAL
WP_1.Display_handler --> EXTERNAL
WP_2.Display_handler --> EXTERNAL
WP_3.Display_handler --> EXTERNAL
WP_number.Display_handler --> Compute_bearing_distance
WP_3.Display_handler --> Compute_bearing_distance
WP_2.Display handler --> Compute_bearing_distance
WP_1.Display_handler --> Compute_bearing_distance
WP_number.Display_handler --> EXTERNAL
Most_recent_position.Display_handler --> EXTERNAl
New_choice.EXTERNAL --> Check_keyboard
Old_time.Compute_position --> Compute_position
Most_recent_position.Compute_position --> Display_handler
Most_recent_position.Compute_position --> Compute_bearing_distance
WP_number.EXTERNAL --> Display_handler
New_time.EXTERNAL --> Compute_position
WP_1.EXTERNAL --> Display_handler
WP_2.EXTERNAL --> Display_handler
WP_3.EXTERNAL --> Display_handler
Present_Position.EXTERNAL --> Display_handler
Course.EXTERNAL --> Display_handler
Speed.EXTERNAL --> Display_handler

DATA STREAM

Bearing	: FLOAT;
Distance	: FLOAT;
Speed	: INTEGER;
Course	: FLOAT;
WP_number	: INTEGER;
WP_3	: POSITION;
WP_2	: POSITION;
WP_1	: POSITION;
Old_Position	: POSITION;
Old_choice	: INTEGER;
New_choice	: INTEGER;
Most_recent_position	: POSITION;

The three lines

```
[1] Old_choice.Check_keyboard --> Check_keyboard
[2] Course.Display_handler --> Compute_position
[3] Present_Position.EXTERNAL --> Display_handler
```

are typical for the possible data streams. [1] represents a state variable and can be read as: there is a data stream called Old_choice originating at operator Check_keyboard and also ending at that operator. [2] is a standard data stream between two operators. [3] shows an external input to the system, a similar format is used for outputs.

The last items needed to completely specify operator INS are potential control constraints for its subcomponents, which have been defined as:

```
CONTROL CONSTRAINTS

  OPERATOR DISPLAY_HANDLER
    PERIOD 1s
  OPERATOR COMPUTE_BEARING_DISTANCE
    PERIOD 1s
  OPERATOR COMPUTE_POSITION
    PERIOD 1s
```

These constraints do not appear in graphic representation, since it only shows maximum execution times. For clarification of an operator the design language includes a description construct.

```
DESCRIPTION

  {This is the root operator. It is composite and consists of the composite operator
  DISPLAY_HANDLER and the atomic operators CHECK_KEYBOARD,
  COMPUTE_BEARING_DISTANCE and COMPUTE_POSITION}

END
```

Since the rest of the development is a repetition of the steps described so far, that work is not presented here. A complete PSDL specification for the system can be found in Appendix E. Operator COMPUTE_POSITION is used on the next page to clarify a certain aspect which might confuse the reader.

```
OPERATOR COMPUTE_POSITION

SPECIFICATION

    INPUT    Speed                   : INTEGER;
             Course                  : FLOAT;
             Old_Position            : POSITION;
             New_time                : TIME;

    OUTPUT   Most_recent_position: POSITION;

    STATE    Old_time                : TIME;

END
```

Part of a complete PSDL implementation of an operator is the TRIGGER CONDITION, which can take on the values BY ALL or BY SOME [Ref. 6:p. 26]. The fact that no TRIGGER CONDITION is used indicates that the default value TRIGGERED BY ALL is used. In the case of operator COMPUTE_POSITION all four inputs SPEED, COURSE, OLD_POSITION and NEW_TIME have to be present to fire the operator.

III. THE FORMAL SOFTWARE ENGINEERING APPROACH

A. PREFACE

The system development will follow the steps as outlined in [Ref. 9] which was summarized in the introduction [see p. 4]. It is assumed, that the reader has familiarized himself with the sequence and purpose of each step. This is a case study aimed at exploring methods for software development and not at creating a system of production quality for operational use, therefore certain aspects of the system such as the concept of 'wind' will be left out of consideration.

B. THE INITIAL PROBLEM STATEMENT

The proposed software system is an Inertial Navigation System (INS) to be used in aircraft. It interacts with the flight directory system. The system must be capable of deriving the present position of the aircraft and provide information about the flight parameters as well as destination data for selected destinations. Additional data needed for aircraft steering must be available.

C. REQUIREMENTS ANALYSIS

1. The System's Environment Model

To create a vocabulary to which all persons involved in the development process can refer and agree a model of the system's environment is built. For this example it is the following:

- The INS will be a software system.
- It will interact with the flight directory system (FDS), the user and the velocity unit (VU).
- The FDS is a device used to steer the aircraft in an automatic mode.
- The VU is the part of the overall system where the aircraft acceleration in all three dimensions is measured and converted into velocities.
- Automatic mode describes the fact that the aircraft is steered by the computer and not by the pilot.

- The present position is the aircraft's position relative to the earth's surface, it is expressed in terms of latitude and longitude.

- Flight parameters are measures of the aircraft's behavior with respect to movement in space consisting of the components course, speed and altitude.

- A destination is a point in space expressed in the same terms as present position.

- Destination data are measures of the relative position of the aircraft to the destination.

- Data for steering the aircraft are those that are needed by the flight directory system to steer the aircraft to the selected destination.

2. Goals and Functions of the System

To derive the high level goals the initial problem statement is used. For the proposed system they are:

G1: The purpose of the INS is to help the aircrew to navigate their aircraft.

G1.1: The system must provide information about the state of the aircraft.

G1.2: The system must calculate destination data for destination positions.

G1.3: The system must provide data necessary to steer the aircraft.

G1.4: The system is supposed to be highly concurrent and prepared for future extensions.

3. Constraints

With the development of every system certain constraints like a fixed budget or delivery dates are associated; which are usually implied by the customer. For our example they are aimed at making this project feasible and suitable for the academic environment.

Implementation Constraints

C1: The system has to be implemented in Ada

C2: The implementation should aim at a high level of concurrency.

C3: The compilers available are

- VERDIX on a SUN workstation
- Meridian AdaVantage on a IBM XT compatible PC
- INTEGRADA on a IBM XT compatible PC

Performance Constraints

C4:The positional data and destination data have to be updated every second.

C5:The system must allow for future extensions.

Resource Constraints

C6:The system must be developed within three month by one person.

4. Refined Goals

Continuing in the development process, the high level goals derived earlier, have to be refined. This is done by identifying the concepts in the high level goals which need to be explained further. The goals are repeated here for easier reference.

G1.1: The system must provide information about the state of the aircraft.

The concept of 'state of the aircraft' needs to be refined; it consists of information about the aircraft's position and its flight parameters. These concepts have been explained in the environment model; therefore the refined goals for G1.1 are:

G1.1.1: The system must provide the aircraft present position.

G1.1.2: The system must provide the aircraft course.

G1.1.3: The system must provide the aircraft speed.

G1.1.4: The system must provide the aircraft altitude.

Another level of refinement is needed to define the units of the above introduced entities and their meanings.

G1.1.1.1: The position consists of latitude and longitude, both measured in degrees(°), Minutes(') and Seconds("). Latitude can take on values from 90° south to 90° north. The range for longitude extends from 180° west to 180° east.

G1.1 2.1: Course is measured in degrees(°), oriented to true north which equals a course of 0°.

G1.1.3.1: Speed is measured in knots(KTS) and can range from 0 to 499KTS.

G1.1.4.1: Altitude is measured in feet(ft) and ranges from 0 to 50000ft.

G1.2: The system must calculate destination data for destination positions

'Destination data' as mentioned in the environment model determine the relative position of the aircraft to a destination position. This relation is expressed in terms of true bearing and distance.

G1.2.1: The system must calculate the true bearing from the aircraft to a destination position.

G1.2.2: The system must calculate the distance from the aircraft to a destination position.

G1.3: The system must provide data necessary to steer the aircraft.

In G1.3 the concept of 'data necessary to steer the aircraft' needs refinement. The environment model mentions that the aircraft can be flown in automatic mode. Consequently the steering data must be those needed to employ that automatic mode. In order for the aircraft to fly to a destination position it needs a direction to fly in, which e.g. can be provided as a bearing relative to the present course. This relative bearing is the difference between the present aircraft course and the true bearing of the aircraft to the destination position. Refinement of G1.3 results in:

G1.3.1: The system must provide true bearing to a destination position.

G1.3.2: The system must provide relative bearing to a destination position.

After defining all these goals the question arises how they can be met; where does all the information to satisfy the goals come from? In this case all the necessary data will be computed inside the INS and the data these computations will be based on will enter the system via its interfaces to the user and the velocity unit, which will be defined in the functional specification.

G1.4: The system is supposed to be highly concurrent and prepared for future extensions.

The goals in G1.4 cannot be refined here, they will be considered in the architectural design stage and in the implementation.

D. FUNCTIONAL SPECIFICATION

Berzins provides procedures and guidelines for the conduct of a functional specification in [Ref.9:p. 3-16]. Each step is quoted here to enable the reader to follow the development process more easily.

STEP 1

"Identify the major subsystems of the proposed software and the user classes and external systems with which the proposed software system will interact."

Using the environment model created earlier, the following entities are identified:

- There will be one INERTIAL_NAVIGATION_SYSTEM software system.
- The system will interact with three external systems, USER, FLIGHT_DIRECTORY_SYSTEM and VELOCITY_UNIT, the latter two being hardware devices.

No subsystems are identified at this time.

STEP 2

"Identify all external interfaces of the proposed subsystems, and make a list of the messages in each interface. Make sure the identified messages correspond to the goal hierarchy, and go over the lists with the customer. Create a SPEC module for each external system, subsystem and interface. Set up the inheritance links between the interfaces and the proposed subsystems."

There are three external systems, one interface for each one is needed. They are named as: user_interface, flight_directory_system_interface and velocity_unit_interface. The definitions given so far are summarized in [Figure 5:p. 21] to insure the proper understanding of the general layout of the proposed system before continuation.

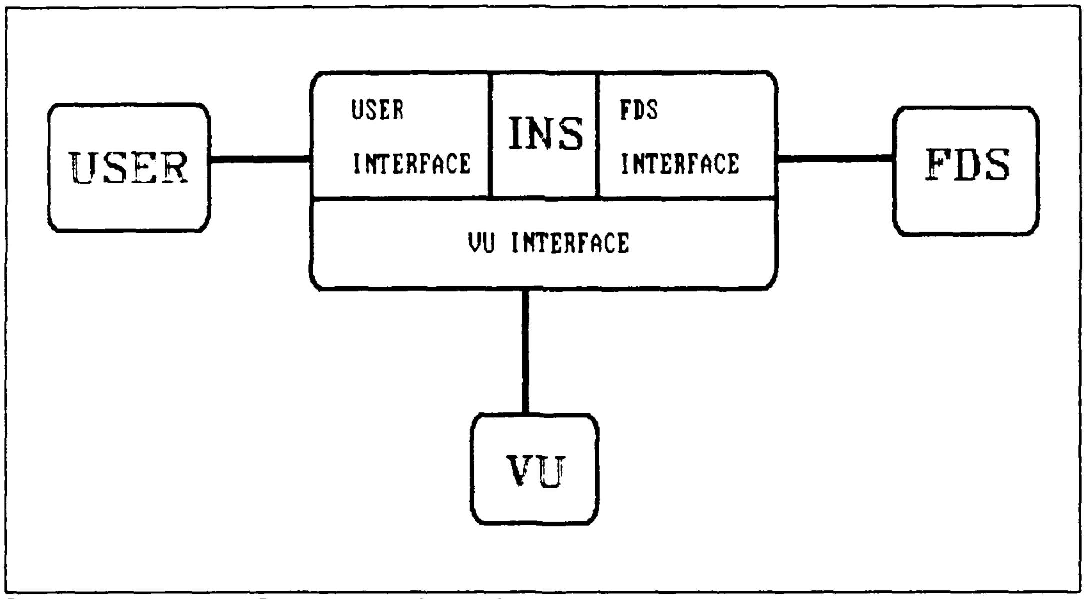

Figure 5: External Systems and Interfaces

To avoid repetition of writing and reading, the lists of messages pertaining to each interface are incorporated into the corresponding SPEC constructs right away. A '?' in a specification marks an entity that is non trivial and needs further refinement in a later stage of the specification process. The resulting specification are shown on the next page:

```
MACHINE inertial_navigation_system
  INHERIT user_interface
  INHERIT flight_directory_system_interface
  INHERIT velocity_unit_interface

  STATE
  INVARIANT true
  INITIALLY true
END


MACHINE flight_directory_system
  STATE ?
  INVARIANT true
  INITIALLY true

  -- The flight_directory_system is a hardware system, therefore it will not be considered
  -- any further in the development process.
END


MACHINE velocity_unit
  STATE ?
  INVARIANT true
  INITIALLY true

  -- The velocity_unit is a hardware system, therefore it will not be considered any
  -- further in the development process.
END


MACHINE user
  STATE ?
  INVARIANT true
  INITIALLY true
END


MACHINE user_interface
  STATE ?
  INVARIANT true
  INITIALLY true

  MESSAGE new_position
    -- Enables the user to enter the coordinates for a new present position into the
    -- system.

  MESSAGE define_waypoint
    -- Enables the user to enter the coordinates for a destination waypoint into the
    -- system.
```

```
  MESSAGE select_waypoint
    -- Enables the user to select one of the waypoints as a destination for computing
    -- destination data from there on.

  MESSAGE display_select
    -- Enables the user to select a data item for display on the screen.
END

MACHINE flight_directory_system_interface
  STATE ?
  INVARIANT true
  INITIALLY true

  MESSAGE relative_bearing_to_a_WP
    -- Requests a relative bearing inertial_navigation_system to a selected waypoint for
    -- steering the aircraft.
END


MACHINE velocity_unit_interface
  STATE ?
  INVARIANT true
  INITIALLY true

  MESSAGE new_velocities
    -- Provides new velocity data to MACHINE ins.
END
```

Since this is an example aiming at exploring the principles of software engineering and not actually develop a complete system, the further development and refinement will not be done for all components but only for those, which give good examples for what is supposed to be done in each step or are suitable to introduce new concepts. For step three the MACHINE user_interface has been chosen.

STEP 3

"For each interface, write down a skeleton specification for all of the messages. Choose names for all messages, exceptions and message components, and identify the data type of each message component. Identify any new abstract data types needed, and create SPEC modules for them. When all of the components have been identified, make an initial estimate of how much effort it will take to build the system."

Step three yields the following result for MACHINE user_interface, where the comments relate to the corresponding goal, developed in the requirements analysis:

```
MACHINE user_interface
  STATE ?
  INVARIANT true
  INITIALLY true

  MESSAGE new_position (p: position)                              --G1.1.1
    TRANSITION ?

  MESSAGE define_waypoint (waypoint: position, wp_number: waypoint_number_range)
   -- G1.2
    WHEN ?
      TRANSITION ?
    OTHERWISE REPLY invalid_waypoint_number

  MESSAGE select_waypoint (wp_number: integer)                    --G1.2
    TRANSITION ?

  MESSAGE display_select (display_selection: display_option)      --G1.1, G1.2
    TRANSITION ?

  TEMPORAL update_display WHERE PERIOD ?
    SEND ?

END
```

A TEMPORAL clause has been introduced here to represent the time dependant behavior of the interface. It will be elaborated later on.

No abstract data types are identified at this time, since no other operations than input and output are performed on either of the data types position, real and integer.

STEP 4

"Invent conceptual models for each machine and type. Develop the invariants and initial conditions, and define the concepts needed to specify them. Check the consistency of the interfaces, and make any adjustments needed."

Before the INVARIANT and INITIALLY conditions can be discussed, it is necessary to elaborate the STATE of the interface. It is to contain the following entities:

- Present position
- Course
- Speed
- Altitude

- Waypoints 1 to 3 (From here on the system is supposed to be able to handle up to three waypoints)
- Current_waypoint_number
- Display_selection

Since the components in the STATE can take on only defined values, e.g. display_selection can take on only those values enumerated in type display_option, and there are no unallowed interactions between the components in the state, INVARIANT is true for all possible STATES.

All components in STATE are initialized before the user takes control over the program. The refined specification for MACHINE user_interface:.

```
MACHINE user_interface
  STATE (     present_position    : position,
              course              : bearing_range,
              speed               : speed_range,
              altitude            : altitude_range,
              waypoint_1          : position,
              waypoint_2          : position,
              waypoint_3          : position,
              current_wp_number   : waypoint_number_range,
              display_selection   : display_option )

  INVARIANT true

  INITIALLY   present_position    = [latitude::0.0,longitude::0.0],
              course              = 0.0,
              speed               = 0,
              altitude            = 0,
              waypoint_1          = [latitude::0.0,longitude::0.0],
              waypoint_2          = [latitude::0.0,longitude::0.0],
              waypoint_3          = [latitude::0.0,longitude::0.0],
              current_wp_number   = 1,
              display_selection   = present_position_choice

  MESSAGE new_position (p: position)
    TRANSITION ? -- update coordinates for present_position

  MESSAGE define_waypoint (waypoint: position, wp_number: waypoint_number_range)
    WHEN ? -- distinguish between waypoints
      TRANSITION ? -- update coordinates for a waypoint
    OTHERWISE REPLY EXCEPTION invalid_waypoint_number

  MESSAGE select_waypoint (wp_number: integer)
    TRANSITION ? -- update the waypoint selection
```

```
  MESSAGE display_select (display_selection: display_option)
    TRANSITION ? -- update display choice

  TEMPORAL update_display WHERE PERIOD ?
    SEND ?

  CONCEPT position: type
    WHERE ?

  CONCEPT bearing_range: type
    WHERE ?

  CONCEPT speed_range: type
    WHERE ?

  CONCEPT altitude_range: type
    WHERE ?

  CONCEPT waypoint_number_range: type
    WHERE ?

  CONCEPT distance_range: type
    WHERE ?

  CONCEPT display_option: type
    WHERE ?
END
```

STEP 5

"Develop the WHEN, WHERE and TRANSACTION clauses for each message and identify the concepts needed to specify them. Refine the invariants as needed. Determine IMPORT, EXPORT relations for shared concepts and create definition skeletons for each concept. The definition skeletons should define the types of inputs and outputs for each concept, and should have an informal description of the concept."

STEP 6

"Write formal definitions for concepts, identifying any necessary lower level concepts, and writing definition skeletons for them. Continue until all concepts have been defined in terms of built-in or available components. Check the internal consistency of the entire specification, and resolve any conflicts."

Steps five and six are combined. All the WHERE and WHEN clauses that were marked by a '?' in the previous step are elaborated here. The result is shown on the next page.

```
MESSAGE new_position (p: position)
  TRANSITION present_position = p

MESSAGE define_waypoint (waypoint: position, wp_number: waypoint_number_range)
  WHEN current_wp_number = 1
    TRANSITION waypoint_1 = waypoint
  WHEN current_wp_number = 2
    TRANSITION waypoint_2 = waypoint
  WHEN current_wp_number = 3
    TRANSITION waypoint_3 = waypoint
  OTHERWISE -- no other choice possible due to type restriction for wp_number

MESSAGE select_waypoint (wp_number: integer)
  TRANSITION current_wp_number = wp_number

MESSAGE display_select (display_selection: display_option)
  TRANSITION *display_selection = display_selection

TEMPORAL update_display WHERE PERIOD = (1 second)
  WHEN display_selection = present_position_choice
    SEND display(p: position) TO user
      WHERE p = present_position
  WHEN display_election = course_choice
    SEND display(c:bearing) TO user
      WHERE c = course
  WHEN display_selection = speed_choice
    SEND display(s:integer) TO user
      WHERE s = speed
  WHEN display_selection = altitude_choice
    SEND display(a:altitude_range) TO user
      WHERE a = altitude
  WHEN display_selection = waypoint_choice
    SEND (w: position) TO user
      WHERE IF current_waypoint_number = 1 THEN w = waypoint_1
                 ELSE IF current_waypoint_number = 2 THEN w = waypoint_2
                 ELSE w = waypoint_3
                 FI
  WHEN display_selection = true_bearing_to_a_wp_choice
    SEND (t: bearing_range) TO user
      WHERE t = true_bearing(present_position, waypoint:: position)

  WHEN display_selection = distance_to_a_wp_choice
    SEND (d: distance_range) TO user
      WHERE d = distance(present_position, waypoint:: position)
  OTHERWISE -- no other choice possible due to type restriction for
                 -- display_selection

CONCEPT position: type
  WHERE position = TUPLE{latitude:: lat_range, longitude:: lon_range}
  -- The meaning of type position is explained in G1.1.1.1.
```

```
CONCEPT bearing_range: type
  WHERE subtype(bearing_range, real) & ALL(b: bearing_range:: 0.0<=b<360.0)
  -- A compass rose has values from 0.0 to 360.0 degrees

CONCEPT speed_range: type
  WHERE subtype(speed_range, integer) & ALL(s:speed_range:: 0<=s<500)
  -- Maximum speed allowed is 500 kts

CONCEPT altitude_range: type
  WHERE subtype(altitude_range, integer) & ALL(a:altitude_range:: 0<=a<=50000)
  -- Maximum altitude allowed is 50000 feet

CONCEPT waypoint_number_range: type
  WHERE subtype(waypoint_number_range, integer) &
          ALL(w:waypoint_number_range:: 1<=w<=3)
  -- Only three waypoints are allowed

CONCEPT distance_range: type
  WHERE subtype(distance_range, real) & ALL(d:distance_range:: 0.0<=d<=10800.0)
  -- 10800 is the maximum number of nautical miles between two points on the
  -- earth's surface it is equal to half its circumference.

CONCEPT display_option: type
  WHERE display_option = enumeration {  present_position_choice,
                                        course_choice,
                                        speed_choice,
                                        altitude_choice,
                                        waypoint_choice,
                                        true_bearing_to_a_wp_choice,
                                        distance_to_a_wp_choice}
  -- The display_option is a way for the user to control, which data item is displayed
  -- on the screen.

CONCEPT distance(present_position waypoint: position)
  VALUE (d: distance_range)
  -- uses a formula from spherical geometry to calculate the distance between two
  -- points on earth's surface and expresses it in terms of distance_range

CONCEPT bearing(present_position waypoint: position)
  VALUE (b: bearing_range)
  -- uses a formula from spherical geometry to calculate the bearing between two
  -- points on earth's surface and expresses it in terms of bearing_range

CONCEPT lat_range: type
  WHERE subtype(lat_range, real) & ALL(l: lat_range:: -90.0<=l<=90.0)

CONCEPT lon_range: type
  WHERE subtype(lon_range, real) & ALL(l: lon_range:: -180.0<=l<=180.0)
END
```

The above is the complete abstract functional specification for MACHINE user_interface and marks the end of the mechanical development, since the remainder would be a repetition of the used methods of refinement. As a first result of this work it shall be mentioned here, that this kind of process is not suitable for a manual approach. It will only be feasible for large software system after automated tools have been developed, which aid the designer/analyst in the process, e.g. a syntax checker is already available and was used to verify the syntactical correctness of the specification; a typechecker and a syntax directed editor are currently under development.

E. ARCHITECTURAL DESIGN

The architectural design for the INS system does not have to be developed using the SPEC language, since this step was already accomplished in the PSDL development, for a review see [Figure 2:p 7], [Figure 3:p 10] and [Figure 4:p. 12]. The design is ready to be implemented at this stage.

IV. IMPLEMENTATION

A. PREFACE

Up to this point we have explored methods to create software in an automated fashion. Since not all tools are operational yet, the implementation of the INS system was done in the traditional 'manual' way. This approach is worthwhile because it gives a good bases for future work. When all the tools become available, a test case will already be available which can be used to compare automatically and manually produced software. Even though the implementation was done manually, the author tried to stay as close to the development work done so far as possible. Parts of the code for the INS system are shown in this chapter, for the full implementation consult Appendix B. Actual code is typed in bold face to visually separate it from the text.

B. COMPILER

The implementation was done using two compilers:

1. INTEGRADA

The system runs on an IBM XT personal computer and was used to develop subcomponents to be integrated into the overall system at a later stage.

INTEGRADA is not only a compiler, but a development environment, providing an editor which can be used as a normal programmer's editor or as a syntax or language directed editor. This was considered useful, since the Ada language is very rich in its available constructs, and the syntax generation capability saved a lot of time in consulting the Ada language reference manual (ALRM) [Ref. 10] and other literature.

Another feature of INTEGRADA is the pretty printer which allows the user to format the source code in several ways. The option 'Program Structure' is very helpful for debugging purposes and the option 'MIL STD 1815 A' [Ref. 10] was used after all the source code had reached its final stage to format the documents in a format as described in the ALRM and that is accepted in the Ada community.

2. VERDIX

The target machine for the final product was a SUN workstation, the compiler available on this system is the VERIDX Ada compiler Version 5.5 for the SUN 3. In contrast to INTEGRADA this compiler is a stand alone version, not an environment, although some tools are provided with the system. To be mentioned are the source level debugger which was very helpful in the implementation phase and the pretty printer.

C. CONCURRENCY AND EXTENSIBILITY

During the formal requirements analysis the goal G1.4 was derived (see also p. 17)

G1.4: The system is supposed to be highly concurrent and prepared for future extensions.

This goal was realized in part during the decomposition of the prototype approach by dividing the system into four separate processes, which can be executed concurrently (see also Figure 3:p. 10). In the implementation these processes are implemented as four independent tasks, whose skeletons are shown on the next page.

```
procedure INS is
   •
   •
  task CHECK_KEYBOARD is
      •
      •
  end CHECK_KEYBOARD;

  task COMPUTE_POSITION is
      •
      •
  end COMPUTE_POSITION;

  task COMPUTE_BEARING_DISTANCE is
      •
      •
  end COMPUTE_BEARING_DISTANCE;

  task DISPLA._HANDLER is
      •
      •
  end DISPLAY_HANDLER;
end ins;
```

This approach has the inherent problem of data integrity. Some of the tasks operate on the same data elements and the question is, how to ensure that no two tasks try to reference and update the same data element at the same time, a problem which is new in multitasking environments, where a program is no longer a set of instructions which are executed in sequence.

A solution was found in an algorithm presented in [Ref. 11]. It uses a task with two entries, one entry allows data to be written to a buffer, the other one allows reading from that buffer. Since the two 'accept' statements are incorporated in a select statement, only one of them can be executed at a time, thereby ensuring data integrity. This data buffer was implemented as a generic package containing a task type. Since the package is generic, it can be instantiated for different data types, the task type allows the creation of several instances of the same type. The accessibility of the data also provides for future extensions to the system. The actual source code used in the INS system is shown on the next page.

```
generic

  type ITEM_TYPE is private;

  package DATA_STORAGE is

    task type BUFFER is
      entry STORE(ITEM  : in ITEM_TYPE);
      entry RECALL(ITEM : out ITEM_TYPE);
    end BUFFER;
  end DATA_STORAGE;

  package body DATA_STORAGE is

    task body BUFFER is
      DATUM : ITEM_TYPE;
    begin
      loop
        select
          accept STORE (ITEM : in ITEM_TYPE) do
            DATUM := ITEM;
          end STORE;
        or
          accept RECALL (ITEM : out ITEM_TYPE) do
            ITEM := DATUM;
          end RECALL;
        end select;
      end loop;
    end BUFFER;
  end DATA_STORAGE;
```

To accommodate all buffers necessary for the INS system nine tasks which serve as data buffers were instantiated.

A drawback of the multitasking concept was found during the development of the input facilities. Due to the underlying operating system (UNIX) it was necessary to serialize the two tasks CHECK_KEYBOARD and DISPLAY_HANDLER, which doesn't affect the functionality of the overall system nor its efficiency or speed. However the implementation is very sytem dependant for this part, which degrades portabibilty. Since problems of this nature were not the main subject for this research they were not investigated any further, which might have resulted in other solutions.

D. TIMING CONSTRAINTS

During the prototype development, time constraints were placed on some of the operators. To show the principle of implementing such constraints, task COMPUTE_BEARING_AND_DISTANCE is discussed.

```
task body COMPUTE_BEARING_DISTANCE is
  •
  •
begin
    •
    •
 loop
    TASK_START := CLOCK;
  -- starts a stopwatch local to this task
    •
    •
  -- statements to execute the necessary computations
    •
    •
    TASK_DONE := CLOCK;
  -- stops the stopwatch
    delay 1.0 - (TASK_DONE - TASK_START);
  -- pauses the task
  end loop;
end COMPUTE_BEARING_DISTANCE;
```

When the task enters the loop, a stopwatch local to this task is started. After all the computations are executed and just before the end of the loop the stopwatch is stopped. The task is then delayed for a period of one second minus the time it took to execute the loop, thereby creating a repetition time or period of one second for the loop. Should the difference be negative, which indicates that the loop needed more than one second to executg, the task will not be delayed and the next loop execution will start right away. According to the Ada standard, this does not necessarily mean the next loop execution starts exactly one second after the last one, but that the task is put in a 'ready' state, waiting for resources. When the necessary resources are available, the task is put into the 'running' state and execution starts.

E. PACKAGING

The system was divided into a main program and four packages. Two of the four packages are generic and were instantiated in multiple instances.

- procedure INS
- package NAVUTIL
- generic package FLOATING_POINT_UTILITIES
- package TERMINAL
- generic package DATA_STORAGE

Packages NAVUTIL, FLOATING_POINT_UTILITIES and TERMINAL represent collections of resources, package DATA_STORAGE implements a buffer data type. In addition to these user defined packages five additional packages supplied with the compiler were used:

- package TEXT_IO
- package MATH
- package CURSES
- package IOCTL
- package SYSTEM

1. Generic package DATA_STORAGE

This package was already discussed in Chapter IV.C. Here an example of its use is given. A navigation system needs the capability to store a geographical position, consequently a buffer was instantiated for this purpose:

```
package POSITION_STORAGE is new DATA_STORAGE(POSITION);
```

where POSITION is a user defined record data type. This makes a task type BUFFER available for data type POSITION. Then a variable of that data type is declared:

```
WP_BUFFER : array (0 .. MAX_WAYPOINTS) of POSITION_STORAGE.BUFFER;
```

The position is stored in one of the array elements. An example of its usage is the task for computing the PRESENT_POSITION shon below.

```
task body COMPUTE_POSITION is
 •
 •
begin
   •
   •
  WP_BUFFER(0).RECALL(PRESENT_POSITION);
 -- retrieves the old PRESENT_POSITION from its buffer
   •
   •
 -- statements to do the calculation
   •
   •
  WP_BUFFER(0).STORE(PRESENT_POSITION);
 -- stores the new PRESENT_POSITION into its buffer
   •
   •
end COMPUTE_POSITION;
```

2. Package TERMINAL

Terminal is the only package that contains hardware dependant code, hence the specification and the body were located in separate files. If the system is to be ported to another system, which has different terminal capabilities, the body of package TERMINAL is the only part that needs to be recoded and recompiled. The current version contains options to run the system on a SUN workstation or a VT 100 terminal.

3. Generic package FLOATING_POINT_UTILITIES

The FLOATING_POINT_UTILITIES package contains some mathematical functions not provided in the standard math library. Most of the algorithms were taken from [Ref. 12]. The functions listed below.

```
functio- INTEGER_PART
function REAL_PART
function FLOOR
function CEILING
function IS_POSITIVE
function IS_NEGATIVE
function INT_TO_CHAR
function CHAR_TO_INT
```

These functions were primarily used in conjunction with input/output operations, which are all done in string or character format, to allow more control over the screen layout. A sample screen is shown in the user manual in Section IV.F of this thesis.

4. Package NAVUTIL

All the functions used to perform the necessary computations in the INS system are located in this package. It also includes the functions for input and output of navigation specific data.

```
procedure GET_POSITION
procedure GET_SPEED;
procedure GET_COURSE;
procedure DISPLAY_POSITION
procedure BEARING_DISTANCE
procedure UPDATE_POSITION
```

As an example for an input operation procedure GET_COURSE is shown here. The input is supposed to be in the form DDD.D, where D is a digit from '0' to '9'.

```
procedure GET_COURSE is
   •
   •
  begin
    •
    •
   -- read in the string
   GET(COURSE_S);
   -- check for period in the correct place
   if COURSE_S(4) = '.' then
     SUCC1 := TRUE;
   else
     SUCC1 := FALSE;
   end if;
   -- convert string to a variable of type FLOAT
   COURSE_F := FLOAT(CHAR_TO_INT(COURSE_S(1)) * 100 +
   CHAR_TO_INT(COURSE_S(2)) * 10 + CHAR_TO_INT(COURSE_S(3))) +
   FLOAT(CHAR_TO_INT(COURSE_S(5))) * 0.1;
   -- check that value is in range
   if COURSE_F >= 0.0 and COURSE_F < 359.9
     then SUCC2 := SUCC1 and TRUE;
   else
     SUCC2 := FALSE;
   end if;
    •
    •
  end GET_COURSE;
```

The remaining input operations for the system are similar, and differ only in the input string length and the checks to be passed, before an input is accepted as valid. These checks are embedded in loops, which can only be exited on a valid input.

F. USER MANUAL

1. Start Up

Only one file named 'INS' is necessary to run the system, it is invoked without any parameters. The system interacts with the user only via the keyboard. Although some error checking is implemented in the system, some errors are unrecoverable at run time. In such cases program execution has to be aborted by pressing the 'CONTROL' key and the 'C' key at the same time. After an internal start up sequence the user is presented with the screen shown below.

```
                         I N S   S I M U L A T O R
------------------------------------------------------------------------


   LATITUDE   N0000.0                   LONGITUDE W00000.0




------------------------------------------------------------------------
   ENTER / UPDATE                      DISPLAY
   --------------                      --------------

   [1] PRESENT POSITION                [6] PRESENT POSITION
   [2] WAYPOINT                        [7] WAYPOINT
   [3] COURSE                          [8] COURSE / SPEED
   [4] SPEED                           [9] BEARING / DISTANCE
```

The user may now enter a start position. The format for entering the information is always the same as presented on the screen, e.g. to enter the latitude:

- Enter 'N' for north or 'S' for south in upper or lower case letters.

- Enter four digits, two for degrees of latitude and two for minutes of latitude.
- Enter a decimal point.
- Enter one digit for decimal fractional minutes of latitude.

After the start position is entered, the user is prompted to enter course and speed values, then the program takes over control and automatically selects option number [6] (DISPLAY PRESENT POSITION). This marks the end of the start up sequence. The system will continue to display the updated present position until the user selects another choice from the menu, which is continuously displayed on the screen.

2. Run Time Options

Generally an option stays in effect until another one is selected. The system updates the screen once every second as long as it is in one of the DISPLAY options [6] to [9]. In the ENTER / UPDATE options the user can take as much time as he needs to complete an input. The following options are provided:

- ENTER / UPDATE
  - [1] PRESENT POSITION — To enter a present position into the system, behaves as described in the start up section.
  - [2] WAYPOINT — To enter up to three waypoints, numbered 1 to 3. After selection prompts for a waypoint number, then the position can be entered. The default value for all three waypoints is N0000.0 W00000.0.
  - [3] COURSE — To enter a course, which is one of data elements necessary for the system's computations. This is an artificial option, which not be available on an operational system, since COURSE and also SPEED would be provided by other aircraft systems.
  - [4] SPEED — To enter a speed value ranging from 1 to 499 Kts.
  - [5] STEER TO WAYPOINT — To select one of the waypoints as the next destination. Once a waypoint has been selected the bearing and distance calculations refer to this waypoint. The default value is 1.

- DISPLAY
  - [6] PRESENT POSITION — To display the present position of the aircraft.
  - [7] WAYPOINT — To display the coordinates of a waypoint, which has been selected with option [5].
  - [8] COURSE / SPEED — To display the present values for course and speed.
  - [9] BEARING / DISTANCE — To display a true bearing and distance from the aircraft's present position to a waypoint, which has been selected with option [5].

V. CONCLUSIONS

A. THE ADA LANGUAGE

Ada as a programming language is one of the most powerful languages available today, which has good, but also bad attributes associated with it.

1. Object Oriented Programming (OOP)

The constructs available in the language give it characteristics of object oriented programming language. Packages are an example for data abstraction and encapsulation; they enable the programmer to create abstract data types in a true fashion. If private types or even limited private types are used in the implementation, the only operations available for an abstract data type are those defined by the programmer, or in the case of private types additionally the 'assignment' and 'check for equivalence' operation.

A major ingredient of OOP is inheritance. The 'with' statement in Ada allows a a variable or object of a certain type to inherit characteristics, which e.g. might be defined in a package.

2. Strong Typing

Another characteristic, strong typing, is a very important aspect in connection with large software systems, which are, among others, one reason for Ada's existence. Strong typing can make programming a very cumbersome task, since many type conversions may be necessary. On the other hand it far outreaches this disadvantage, when it comes to debugging a program as all programming errors that result in type inconsistencies are detected at compile time already. For languages that support no or almost no static type checking e.g. 'C' this checking must be done at run time. But then the amount of typing errors detected depends on the data on which the program operates. This is one fact that makes 'C', from a software engineering point of view, unsuitable for large software systems.

3. Information Hiding

Information hiding is implemented very well in the Ada language. Good examples of this are the packages provided with the different compilers. The user is only provided with the interface or specification of the packages, which is always the same for a certain package. Whereas the sourcecode for the body, which may be different for each implementation, is usually not accessible.

4. Concurrency

Ada makes multitasking possible only using constructs defined within the language in the form of tasks and other related constructs, like rendezvous and the pragma 'priority'. This should be a good asset in terms of efficiency and performance, however, as of now, no compiler is available for any multi processor system, but that fact should be eliminated by time, since compilers have already been announced for multiprocessor systems.

5. Portability

Portability is a more negative aspect of the Ada language, even though the Ada Joint Programming Office keeps a strict eye on the quality of the available compilers by validating only those compilers which successfully work on a set of test programs. At first glance this should ensure portability. The problem lies in the specification of the language, which is manifested in the ALRM [Ref. 10] and which in some places leaves too much leeway for the implementation of the compiler. The best example is the pragma 'priority' which allows the assignment of relative importance on a set of tasks, thereby controlling their order of execution. The pragma has to implemented in every compiler, however the range of legal values is left to the particular implementation, which results in quite different values. Since not all compilers provide this information in their documentation, a small program to check those values on any compiler, regardless of the documentation is shown below.

```
with text_io;
use text_io;
with system;

procedure prio is

package priority_io is new integer_io(system.priority);
use priority_io;

begin
  new_page;
  put("min value for priority : ");
  put(system.priority'first);
  new_line;
  put("max value for priority : ");
  put(system.priority'last);
  new_line;
end;
```

A test run on three different compilers, which were available at the time of this research produced the following results.

COMPILER	VALUES FOR PRAGMA PRIORITY
AdaVantage Version 2.0	1 .. 20
INTEGRADA Version 4.01	0 .. 0
Verdix Version 5.5	0 .. 99

This is only one example of a deficiency in the language specification.

The next factor contributing to Ada's bad portability is the lack of standard libraries, provided with the compilers. As an example one might expect a package for mathematical functions, which are not included in the language standard. Again when comparing the three above mentioned compilers we have the following picture:

	AdaVantage	INTEGRADA	Verdix Ver5.5
Package name	math_lib	mathlib	math
function ARCTAN(X)	atan(x)	arctan(x)	arctan(x)

6. Hard Real Time Systems

As shown in Chapter IV.D on page 34 the programmer has possibilities to influence the execution timing of a programming unit; the example also showed, that a delay is only a **minimum** waiting period, meaning, that there is no way to tell the maximum waiting time, which is unacceptable in hard real time systems, where deadlines have to be met. This aspect of the language is a separate research area in itself and shall not be exploited any further here. The interested reader can find further information in [Ref. 13, 14, 15, 16, 17].

7. Final Comment

Summarizing the points made above, the Ada language is very powerful and suited for its purpose. The negative points should not be considered as an attempt to detract from that fact, but is an attempt by the author to show some areas where further improvement is needed.

B. SPECIFICATION AND PROTOTYPING

The languages SPEC and PSDL are not for programming purposes. Conceptually they reside at a higher level of abstraction than programming languages. The development team no longer describes a program in terms of HOW to complete a certain task, but by specifying WHAT tasks are to be completed. Due to their complexity and size, large software systems cannot be realized using traditional programming languages and software engineering techniques only. No single person can comprehend a complete system, therefore the need for communication between all people involved in the development of such a system arises. Furthermore it is becoming more and more difficult to prove the correctness of a program, or to do at least some testing to insure its correctness to a certain level. SPEC is one attempt to solve this problem. It is suitable to develop the specification for a program instead of the program itself. Since the language is strictly based on mathematical rules it has the potential to solve the 'proof of correctness' problem or at least bring it closer to a solution.

The problem with all specification languages, SPEC is only one of them, lies in their application. As the small example, developed in Chapter III, shows, specifications grow rapidly and become incomprehendable at the same pace. It is obvious that automated tools are necessary to use SPEC on a production level to keep track of the development stage and to insure the completeness and consistency of a specification. As already mentioned some of those tools are presently under construction. Their development is supported by the mathematical foundation of SPEC, a negative aspect however is the fact that not every specification can be automatically translated into executable code.

A type checker is needed to check that all types used within a specification at different levels of decomposition conform, whereas a syntax directed editor must take care of the completeness and syntactical correctness of all language constructs used. Another very important tool is a development database, which retains the development up to the current stage. This is important to provide the capability to go back and forth between different levels of decomposition.

SPEC addresses the problems of reliability, modifiability and other related problems mentioned in the Introduction. The other main problem areas in software development are cost and feasibility; PSDL is an attempt to cope with them. It aids the development process. After the requirements for a project have been manifested, PSDL can be used to construct a prototype which in the long run will be a piece of executable code. PSDL does not have a mathematical foundation like SPEC, hence it cannot be used to attack the 'correctness' problem.

The tool development for PSDL has proceeded much further than that for SPEC. Even though it is not possible to create an executable prototype without manual interaction at the present time, tools already available are instrumental for the completed system as their application demonstrated in Chapter II.

C. THE COMBINATION OF PSDL AND SPEC

So far SPEC and PSDL have been examined as separate systems. The latest development in the software engineering discipline is marked by DARPA's (Defence Advanced Research Projects Agency) decision to create a language on top of Ada [Ref. 18]. This language is to provide all the capabilities presently designed in SPEC and PSDL. Future emphasis should be placed on the fusion of the two languages combing their capabilities. Care must be taken that the resulting language is not just a superset. which contains the two languages as complete subsets. Overlapping constructs and methods must be eliminated. Once a minimal version of the system is operational. it can be used to improve on itself, which should speed up the development dramatically.

APPENDIX A. INS SPECIFICATION IN PROTOTYPE DESCRIPTION LANGUAGE (PSDL)

```
OPERATOR INS

SPECIFICATION

   INPUT  Present_Position          : POSITION;
          Course                    : FLOAT;
          Speed                     : INTEGER;
          WP_1                      : POSITION;
          WP_2                      : POSITION;
          WP_3                      : POSITION;
          WP_number                 : INTEGER;
          New_time                  : TIME;
          New_choice                : INTEGER;

   OUTPUT Present_Position          : POSITION;
          Course                    : FLOAT;
          Speed                     : INTEGER;
          WP_1                      : POSITION;
          WP_2                      : POSITION;
          WP_3                      : POSITION;
          WP_number                 : INTEGER;
          Bearing                   : FLOAT;
          Distance                  : FLOAT;

END

IMPLEMENTATION GRAPH

   Old_choice.Check_keyboard --> Check_keyboard
   Old_choice.Check_keyboard --> Display_handler
   New_choice.Check_keyboard --> Display_handler
   Bearing.Compute_bearing_distance --> Display_handler
   Distance.Compute_bearing_distance --> Display_handler
   Speed.Display_handler --> EXTERNAL
   Speed.Display_handler --> Compute_position
   Course.Display_handler --> EXTERNAL
   Course.Display_handler --> Compute_position
   Old_Position.Display_handler --> Compute_position
   Bearing.Display_handler --> EXTERNAL
   Distance.Display_handler --> EXTERNAL
   WP_1.Display_handler --> EXTERNAL
   WP_2.Display_handler --> EXTERNAL
   WP_3.Display_handler --> EXTERNAL
   WP_number.Display_handler --> Compute_bearing_distance
   WP_3.Display_handler --> Compute_bearing_distance
   WP_2.Display_handler --> Compute_bearing_distance
   WP_1.Display_handler --> Compute_bearing_distance
   WP_number.Display_handler --> EXTERNAL
   Most_recent_position.Display_handler --> EXTERNAL
   New_choice.EXTERNAL --> Check_keyboard
   Old_time.Compute_position --> Compute_position
   Most_recent_position.Compute_position --> Display_handler
   Most_recent_position.Compute_position --> Compute_bearing_distance
   WP_number.EXTERNAL --> Display_handler
   New_time.EXTERNAL --> Compute_position
   WP_1.EXTERNAL --> Display_handler
```

```
   WP_2.EXTERNAL --> Display_handler
   WP_3.EXTERNAL --> Display_handler
   Present_Position.EXTERNAL --> Display_handler
   Course.EXTERNAL --> Display_handler
   Speed.EXTERNAL --> Display_handler

   DATA STREAM

         Bearing                     : FLOAT;
           Distance                  : FLOAT;
           Speed                     : INTEGER;
           Course                    : FLOAT;
           WP_number                 : INTEGER;
           WP_3                      : POSITION;
           WP_2                      : POSITION;
           WP_1                      : POSITION;
           Old_Position              : POSITION;
           Old_choice                : INTEGER;
           New_choice                : INTEGER;
           Most_recent_position      : POSITION;

   CONTROL CONSTRAINTS

         OPERATOR DISPLAY_HANDLER
            PERIOD 1s

         OPERATOR COMPUTE_BEARING_DISTANCE
            PERIOD 1s

         OPERATOR COMPUTE_POSITION
            PERIOD 1s

   DESCRIPTION

         {This is the root operator. It is composite and consists of the
         composite operator DISPLAY_HANDLER and the atomic operators
         CHECK_KEYBOARD, COMPUTE_BEARING_DISTANCE and COMPUTE_POSITION}
END


OPERATOR CHECK_KEYBOARD

SPECIFICATION

   INPUT  New_choice                 : INTEGER;

   OUTPUT Old_choice                 : INTEGER;
          New_choice                 : INTEGER;

   STATE  Old_choice                 : INTEGER INITIALLY 6;

END

IMPLEMENTATION ADA CHECK_KEYBOARD

   {The atomic operator CHECK_KEYBOARD requires visibility to datastreams
   OLD_CHOICE and NEW_CHOICE in IN-}

END
```

```
OPERATOR DISPLAY_HANDLER

SPECIFICATION

   INPUT  Old_choice                  : INTEGER;
          New_choice                  : INTEGER;
          Bearing                     : FLOAT;
          Distance                    : FLOAT;
          Most_recent_position        : POSITION;
          Speed                       : INTEGER;
          Course                      : FLOAT;
          WP_number                   : INTEGER;
          WP_1                        : POSITION;
          WP_2                        : POSITION;
          WP_3                        : POSITION;
          Present_Position            : POSITION;

   OUTPUT Speed                       : INTEGER;
          Course                      : FLOAT;
          Bearing                     : FLOAT;
          Distance                    : FLOAT;
          WP_1                        : POSITION;
          WP_2                        : POSITION;
          WP_3                        : POSITION;
          WP_number                   : INTEGER;
          Old_Position                : POSITION;
          Most_recent_position        : POSITION;

END

IMPLEMENTATION GRAPH

   Present_Position.Enter_present_position --> WP_buffer_0
   WP_1.Enter_waypoint --> WP_buffer_1
   WP_2.Enter_waypoint --> WP_buffer_2
   WP_3.Enter_waypoint --> WP_buffer_3
   Course.Enter_course --> Course_buffer
   Speed.Enter_speed --> Speed_buffer
   WP_number.Enter_steer_to_waypoint --> WP_number_buffer
   Most_Recent_P sition.Display_present_position --> EXTERNAL
   WP_Number.Display_waypoint --> EXTERNAL
   WP_1.Display_waypoint --> EXTERNAL
   WP_2.Display_waypoint --> EXTERNAL
   WP_3.Display_waypoint --> EXTERNAL
   Bearing.Display_bearing_and_distance --> EXTERNAL
   Distance.Display_bearing_and_distance --> EXTERNAL
   WP_Number.Display_bearing_and_distance --> EXTERNAL
   Course.Display_course_and_speed --> EXTERNAL
   Speed.Display_course_and_speed --> EXTERNAL
   Most_Recent_Position.WP_buffer_0 --> Display_present_position
   Most_Recent_Position.WP_buffer_0 --> EXTERNAL
   Old_Position.WP_buffer_0 --> EXTERNAL
   WP_1.WP_buffer_1 --> Display_waypoint
   WP_2.WP_buffer_2 --> Display_waypoint
   WP_3.WP_buffer_3 --> Display_waypoint
   Bearing.Bearing_buffer --> Display_bearing_and_distance
   Distance.Distance_buffer --> Display_bearing_and_distance
   Course.Course_buffer --> Display_course_and_speed
   Speed.Speed_buffer --> Display_course_and_speed
   WP_number.WP_number_buffer --> Display_waypoint
   WP_Number.WP_number_buffer --> Display_bearing_and_distance
   Course.EXTERNAL --> Enter_course
   Speed.EXTERNAL --> Enter_speed
   WP_number.EXTERNAL --> Enter_steer_to_waypoint
```

```
    Bearing.EXTERNAL --> Bearing_buffer
    Distance.EXTERNAL --> Distance_buffer
    Present_Position.EXTERNAL --> Enter_present_position
    Most_Recent_Position.EXTERNAL --> WP_buffer_0
    WP_Number.EXTERNAL --> Enter_waypoint
    WP_1.EXTERNAL --> Enter_waypoint
    WP_2.EXTERNAL --> Enter_waypoint
    WP_3.EXTERNAL --> Enter_waypoint

    DATA STREAM

        Present_Position          : POSITION;
        WP_1                      : POSITION;
        WP_2                      : POSITION;
        WP_3                      : POSITION;
        Course                    : FLOAT;
        Speed                     : INTEGER;
        WP_number                 : INTEGER;
        Most_Recent_Position      : POSITION;
        Bearing                   : FLOAT;
        Distance                  : FLOAT;
        WP_Number                 : INTEGER;

    DESCRIPTION

        {The composite operator DISPLAY_HANDLER CONSISTS of the atomic
        operators ENTER_PRESENT_POSITION, ENTER_WAYPOINT, ENTER_COURSE,
        ENTER_SPEED, ENTER_STEER_TO_WAYPOINT, DISPLAY_PRESENT_POSITION,
        DISPLAY_WAYPOINT, DISPLAY_BEARING_AND_DISTANCE,
        DISPLAY_COURSE_AND_SPEED, WP_BUFFER_0, WP_BUFFER_1, WP_BUFFER_2,
        WP_BUFFER_3, BEARING_BUFFER, DISTANCE_BUFFER, COURSE_BUFFER,
        SPEED_BUFFER and WP_NUMBER_BUFFER. It requires visibility to all data
        streams in INS}

END


OPERATOR COMPUTE_BEARING_DISTANCE

SPECIFICATION

    INPUT  WP_number                : INTEGER;
           WP_3                     : POSITION;
           WP_2                     : POSITION;
           WP_1                     : POSITION;
           Most_recent_position     : POSITION;

    OUTPUT Bearing                  : FLOAT;
           Distance                 : FLOAT;

END

IMPLEMENTATION ADA COMPUTE_BEARING_DISTANCE

    {The atomic operator COMPUTE_BEARING_DISTANCE requires visibility to
    datastreams MOST_RECENT_POSITION, BEARING, DISTANCE, WP_1, WP_2, WP_3 and
    WP_NUMBER in INS}

END
```

```
OPERATOR COMPUTE_POSITION

SPECIFICATION

   INPUT  Speed                    : INTEGER;
          Course                   : FLOAT;
          Old_Position             : POSITION;
          New_time                 : TIME;

   OUTPUT Most_recent_position     : POSITION;

   STATE  Old_time                 : TIME;

END

IMPLEMENTATION ADA COMPUTE_POSITION

   {The atomic operator COMPUTE_POSITION requires visibility to datastreams
   COURSE, SPEED, OLD_POSITION and MOST_RECENT-POSITION in INS}

END


OPERATOR ENTER_PRESENT_POSITION

SPECIFICATION

   INPUT  Present_Position         : POSITION;

   OUTPUT Present_Position         : POSITION;

END

IMPLEMENTATION ADA ENTER_PRESENT_POSITION

   {The atomic operator ENTER_PRESENT_POSITION requires visibility to
   datastream PRESENT_POSITION in DISPLAY_HANDLER}

END


OPERATOR WP_BUFFER_0

SPECIFICATION

   INPUT  Present_Position         : POSITION;
          Most_recent_position     : POSITION;

   OUTPUT Old_Position             : POSITION;
          Most_recent_position     : POSITION;

END

IMPLEMENTATION ADA WP_BUFFER_0

   {The atomic operator WP_BUFFER_0 requires visibility to datastreams
   PRESENT_POSITION, MOST_RECENT_POSITION and OLD_POSITION in DISPLAY_HANDLER}

END
```

```
OPERATOR ENTER_WAYPOINT

SPECIFICATION

    INPUT  WP_number                : INTEGER;
           WP_1                     : POSITION;
           WP_2                     : POSITION;
           WP_3                     : POSITION;

    OUTPUT WP_1                     : POSITION;
           WP_2                     : POSITION;
           WP_3                     : POSITION;

END

IMPLEMENTATION ADA ENTER_WAYPOINT

    {The atomic operator ENTER_WAYPOINT requires visibility to datastreams WP_1,
    WP_2, WP_3 and WP_NUMBER in DISPLAY_HANDLER}

END


OPERATOR WP_BUFFER_1

SPECIFICATION

    INPUT  WP_1                     : POSITION;

    OUTPUT WP_1                     : POSITION;

END

IMPLEMENTATION ADA WP_BUFFER_1

    {The atomic operator WP_BUFFER_1 requires visibility to datastream WP_1 in
    DISPLAY_HANDLER}

END


OPERATOR WP_BUFFER_2

SPECIFICATION

    INPUT  WP_2                     : POSITION;

    OUTPUT WP_2                     : POSITION;

END

IMPLEMENTATION ADA WP_BUFFER_2

    {The atomic operator WP_BUFFER_2 requires visibility to datastream WP_2 in
    DISPLAY_HANDLER}

END
```

```
OPERATOR WP_BUFFER_3

SPECIFICATION

   INPUT  WP_3                        : POSITION;

   OUTPUT WP_3                        : POSITION;

END

IMPLEMENTATION ADA WP_BUFFER_3

   {The atomic operator WP_BUFFER_3 requires visibility to datastream WP_3 in
   DISPLAY_HANDLER}

END


OPERATOR ENTER_COURSE

SPECIFICATION

   INPUT  Course                      : FLOAT;

   OUTPUT Course                      : FLOAT;

END

IMPLEMENTATION ADA ENTER_COURSE

   {The atomic operator ENTER_COURSE requires visibility to datastream COURSE
   in DISPLAY_HANDLER}

END


OPERATOR COURSE_BUFFER

SPECIFICATION

   INPUT  Course                      : FLOAT;

   OUTPUT Course                      : FLOAT;

END

IMPLEMENTATION ADA COURSE_BUFFER

   {The atomic operator COURSE_BUFFER requires visibility to datastream COURSE
   in DISPLAY_HANDLER}

END
```

```
OPERATOR ENTER_SPEED

SPECIFICATION

   INPUT  Speed                        : INTEGER;

   OUTPUT Speed                        : INTEGER;

END

IMPLEMENTATION ADA ENTER_SPEED

   {The atomic operator ENTER_SPEED requires visibility to datastream SPEED in
   DISPLAY_HANDLER}

END


OPERATOR SPEED_BUFFER

SPECIFICATION

   INPUT  Speed                        : INTEGER;

   OUTPUT Speed                        : INTEGER;

END

IMPLEMENTATION ADA SPEED_BUFFER

   {The atomic operator SPEED_BUFFER requires visibility to datastream SPEED in
   DISPLAY_HANDLER}

END


OPERATOR ENTER_STEER_TO_WAYPOINT

SPECIFICATION

   INPUT  WP_number                    : INTEGER;

   OUTPUT WP_number                    : INTEGER;

END

IMPLEMENTATION ADA ENTER_STEER_TO_WAYPOINT

   {The atomic operator ENTER_STEER_TO_WAYPOINT requires visibility to
   datastream WP_NUMBER in DISPLAY_HANDLER}

END
```

```
OPERATOR WP_NUMBER_BUFFER

SPECIFICATION

   INPUT  WP_number                : INTEGER;

   OUTPUT WP_number                : INTEGER;

END

IMPLEMENTATION ADA WP_NUMBER_BUFFER

   {The atomic operator WP_NUMBER_BUFFER requires visibility to datastream
   WP_NUMBER in DISPLAY_HANDLER}

END


OPERATOR DISPLAY_PRESENT_POSITION

SPECIFICATION

   INPUT  Most_recent_position     : POSITION;

   OUTPUT Most_recent_position     : POSITION;

END

IMPLEMENTATION ADA DISPLAY_PRESENT_POSITION

   {The atomic operator DISPLAY_PRESENT_POSITION requires visibility to
   datastream MOST_RECENT_POSITION in DISPLAY_HANDLER}

END


OPERATOR DISPLAY_WAYPOINT

SPECIFICATION

   INPUT  WP_number                : INTEGER;
          WP_1                     : POSITION;
          WP_2                     : POSITION;
          WP_3                     : POSITION;

   OUTPUT WP_1                     : POSITION;
          WP_2                     : POSITION;
          WP_3                     : POSITION;
          WP_number                : INTEGER;

END

IMPLEMENTATION ADA DISPLAY_WAYPOINT

   {The atomic operator DISPLAY_WAYPOINT requires visibility to datastreams
   WP_1, WP_2, WP_3 and WP_NUMBER in DISPLAY_HANDLER}

END
```

```
OPERATOR DISPLAY_COURSE_AND_SPEED

SPECIFICATION

   INPUT  Course                        : FLOAT;
          Speed                         : INTEGER;

   OUTPUT Course                        : FLOAT;
          Speed                         : INTEGER;

END

IMPLEMENTATION ADA DISPLAY_COURSE_AND_SPEED

   {The atomic operator DISPLAY_COURSE_AND_SPEED requires visibility to
   datastreams COURSE and SPEED in DISPLAY_HANDLER}

END


OPERATOR BEARING_BUFFER

SPECIFICATION

   INPUT  Bearing                       : FLOAT;

   OUTPUT Bearing                       : FLOAT;

END

IMPLEMENTATION ADA BEARING_BUFFER

   {The atomic operator BEARING_BUFFER requires visibility to datastream
   BEARING in DISPLAY_HANDLER}

END


OPERATOR DISPLAY_BEARING_AND_DISTANCE

SPECIFICATION

   INPUT  Bearing                       : FLOAT;
          Distance                      : FLOAT;
          WP_number                     : INTEGER;

   OUTPUT Bearing                       : FLOAT;
          Distance                      : FLOAT;
          WP_number                     : INTEGER;

END

IMPLEMENTATION ADA DISPLAY_BEARING_AND_DISTANCE

   {The atomic operator DISPLAY_BEARING_AND_DISTANCE  requires visibility to
   datastreams BEARING, DISTANCE and WP_NUMBER in DISPLAY_HANDLER}

END
```

```
OPERATOR DISTANCE_BUFFER

SPECIFICATION

   INPUT  Distance                        : FLOAT;

   OUTPUT Distance                        : FLOAT;

END

IMPLEMENTATION ADA DISTANCE_BUFFER

   {The atomic operator DISTANCE_BUFFER requires visibility to datastream
   DISTANCE in DISPLAY_HANDLER}

END
```

APPENDIX B. ADA SOURCE CODE LISTING

```
--------------------------------------------------------------------------------
-- UNIT_NAME            | ins.a
-- CSCI_NAME
-- UNIT_DESCRIPTION
--
--
-- UNIT_SPS_REFERENCE
-- UNIT_CALLING_SEQUENCE
-- EXTERNAL_UNITS_CALLED
-- INPUTS
-- OUTPUTS
-- CREATED              | 23 January 1989
-- AUTHOR               | herbert guenterberg
-- DATE ------------ AUTHOR -------- REVISION # -- PR # -----TITLE -----------
--------------------------------------------------------------------------------
-- This is the main program for the ins-simulator. Compilation sequence:
--
--         Term_s.a,
--         Term_b.a,
--         Data_sto.a,
--         Mathutil.a,
--         Navutil.a,
--         Ins.a
--
-- To link on a UNIX based system with a VERDIX compiler:
--
--        a.ld -o ins ins -ltermcap -lcurses
---------------------------------------------------------------------------------

with TEXT_IO;
use TEXT_IO;
with TERMINAL;
use TERMINAL;
with NAVUTIL;
use NAVUTIL;
with CALENDAR;
use CALENDAR;
with FLOATING_POINT_UTILITIES;

procedure INS is

  package FLOAT_UTIL is new FLOATING_POINT_UTILITIES(FLOAT);
  use FLOAT_UTIL;

  package INT_IO is new INTEGER_IO(INTEGER);
  use INT_IO;

-- initialization of variables

  INITIAL_POSITION : POSITION := (0.0, 0.0);
  INITIAL_COURSE : FLOAT := 0.0;
  INITIAL_SPEED : INTEGER := 0;
  INITIAL_BEARING : FLOAT := 0.0;
  INITIAL_DISTANCE : FLOAT := 0.0;
  INITIAL_WF : INTEGER := 1;
```

```
-- task declarations; names are selfexplanatory for each task

  task CHECK_KEYBOARD is
    entry START;
    entry STOP;
    entry CONTINUE;
  end CHECK_KEYBOARD;
  task COMPUTE_POSITION is
    entry START;
  end COMPUTE_POSITION;

  task COMPUTE_BEARING_DISTANCE is
    entry START;
  end COMPUTE_BEARING_DISTANCE;

  task DISPLAY_HANDLER is
    entry MAKE_CHOICE(CHOICE : in CHARACTER);
  end DISPLAY_HANDLER;

-- task bodies

  task body CHECK_KEYBOARD is
    NEW_CHOICE : CHARACTER := '6';
    OLD_CHOICE : CHARACTER := '6';
    TASK_START : TIME;
    TASK_DONE : TIME;
  begin
    accept START do
      DISPLAY_HANDLER.MAKE_CHOICE(NEW_CHOICE);
      accept STOP;
      accept CONTINUE;
      SPECIAL_IO;
    end START;
    loop
      TASK_START := CLOCK;
      if KEY_PRESSED then
        GET(NEW_CHOICE);
        if NEW_CHOICE > '0' and NEW_CHOICE <= '9' then
          if NEW_CHOICE > '0' and NEW_CHOICE < '6' then
            CLEAR_LINE(3, 7);
            NORMAL_IO;
          end if;
          DISPLAY_HANDLER.MAKE_CHOICE(NEW_CHOICE);
          accept STOP;
          accept CONTINUE;
          if NEW_CHOICE > '0' and NEW_CHOICE < '6' then
            CLEAR_LINE(3, 7);
            SPECIAL_IO;
          end if;
        end if;
      end if;
      if NEW_CHOICE < '6' then
        DISPLAY_HANDLER.MAKE_CHOICE(OLD_CHOICE);
        else
          DISPLAY_HANDLER.MAKE_CHOICE(NEW_CHOICE);
      end if;
      accept STOP;
      accept CONTINUE;
      if NEW_CHOICE > '5' then
        OLD_CHOICE := NEW_CHOICE;
      end if;
      TASK_DONE := CLOCK;
      delay 1.0 - (TASK_DONE - TASK_START);
```

```
  end loop;
end CHECK_KEYBOARD;

task body COMPUTE_POSITION is
  ACTUAL_TIME : TIME;
  OLD_TIME : TIME;
  INTERVAL : DURATION := 0.0;
  PRESENT_POSITION : POSITION;
  TEMP_COURSE : FLOAT;
  TEMP_SPEED : FLOAT;
  INT_SPEED : INTEGER;
  TASK_START : TIME;
  TASK_DONE : TIME;
begin
  accept START do
    OLD_TIME := CLOCK;
  end START;
  loop
    TASK_START := CLOCK;
    ACTUAL_TIME := CLOCK;
    INTERVAL := ACTUAL_TIME - OLD_TIME;
    OLD_TIME := ACTUAL_TIME;
    WP_BUFFER(0).RECALL(PRESENT_POSITION);
    COURSE_BUFFER.RECALL(TEMP_COURSE);
    SPEED_BUFFER.RECALL(INT_SPEED);
    TEMP_SPEED := FLOAT(INT_SPEED);
    UPDATE_POSITION(INTERVAL, PRESENT_POSITION, TEMP_COURSE,
      TEMP_SPEED);
    WP_BUFFER(0).STORE(PRESENT_POSITION);
    TASK_DONE := CLOCK;
    delay 1.0 - (TASK_DONE - TASK_START);
  end loop;
end COMPUTE_POSITION;

task body COMPUTE_BEARING_DISTANCE is
  PRESENT_POSITION : POSITION;
  TARGET_POSITION : POSITION;
  TEMP_BEARING : FLOAT;
  TEMP_DISTANCE : FLOAT;
  WP_NO : WAYPOINT_RANGE;
  TASK_START : TIME;
  TASK_DONE : TIME;
begin
  accept START;
  loop
    TASK_START := CLOCK;
    WP_NUMBER_BUFFER.RECALL(WP_NO);
    WP_BUFFER(0).RECALL(PRESENT_POSITION);
    WP_BUFFER(WP_NO).RECALL(TARGET_POSITION);
    BEARING_DISTANCE(PRESENT_POSITION, TARGET_POSITION,
      TEMP_BEARING, TEMP_DISTANCE);
    BEARING_BUFFER.STORE(TEMP_BEARING);
    DISTANCE_BUFFER.STORE(TEMP_DISTANCE);
    TASK_DONE := CLOCK;
    delay 1.0 - (TASK_DONE - TASK_START);
  end loop;
end COMPUTE_BEARING_DISTANCE;
```

```
task body DISPLAY_HANDLER is
  OLD_CHOICE : CHARACTER := '1';
  NEW_CHOICE : CHARACTER := '6';

  procedure ENTER_PRESENT_POSITION is
  begin
    CHECK_KEYBOARD.STOP;
    GOTOXY(DR - 3, C1);
    PUT("ENTERING PRESENT POSITION");
    GET_POSITION(0);
    CHECK_KEYBOARD.CONTINUE;
  end ENTER_PRESENT_POSITION;

  procedure ENTER_WAYPOINT is
    WP_NO : INTEGER := MAX_WAYPOINTS + 1;
  begin
    CHECK_KEYBOARD.STOP;
    GOTOXY(DR - 3, C1);
    PUT("ENTERING WAYPOINT NO:");
    while WP_NO > MAX_WAYPOINTS loop
      GOTOXY(DR, C1);
      PUT("ENTER A WAYPOINT NUMBER : ");
      GET(WP_NO);
    end loop;
    GOTOXY(DR - 3, C2);
    PUT(INT_TO_CHAR(WP_NO));
    GET_POSITION(WP_NO);
    CHECK_KEYBOARD.CONTINUE;
  end ENTER_WAYPOINT;

  procedure ENTER_COURSE is
  begin
    CHECK_KEYBOARD.STOP;
    GET_COURSE;
    CHECK_KEYBOARD.CONTINUE;
  end ENTER_COURSE;

  procedure ENTER_SPEED is
  begin
    CHECK_KEYBOARD.STOP;
    GET_SPEED;
    CHECK_KEYBOARD.CONTINUE;
  end ENTER_SPEED;

  procedure ENTER_STEER_TO_WAYPOINT is
    WP_NO : INTEGER := MAX_WAYPOINTS + 1;
  begin
    CHECK_KEYBOARD.STOP;
    while WP_NO > MAX_WAYPOINTS loop
      GOTOXY(DR, C1);
      PUT("ENTER THE TARGET WAYPOINT NUMBER : ");
      GET(WP_NO);
    end loop;
    WP_NUMBER_BUFFER.STORE(WP_NO);
    CHECK_KEYBOARD.CONTINUE;
  end ENTER_STEER_TO_WAYPOINT;
```

```
procedure DISPLAY_PRESENT_POSITION is
  PRESENT_POSITION : POSITION;
begin
  CHECK_KEYBOARD.STOP;
  CLEAR_LINE(DR, 1);
  WP_BUFFER(0).RECALL(PRESENT_POSITION);
  DISPLAY_POSITION(PRESENT_POSITION);
  GOTOXY(DR, C2 + 20);
  PUT("PRESENT POSITION");
  CHECK_KEYBOARD.CONTINUE;
end DISPLAY_PRESENT_POSITION;

procedure DISPLAY_WAYPOINT is
  WP_NO : INTEGER;
  WAYPOINT : POSITION;
begin
  CHECK_KEYBOARD.STOP;
  WP_NUMBER_BUFFER.RECALL(WP_NO);
  CLEAR_LINE(DR, 1);
  WP_BUFFER(WP_NO).RECALL(WAYPOINT);
  DISPLAY_POSITION(WAYPOINT);
  GOTOXY(DR, C2 + 20);
  PUT("WAYPOINT ");
  PUT(INT_TO_CHAR(WP_NO));
  CHECK_KEYBOARD.CONTINUE;
end DISPLAY_WAYPOINT;

procedure DISPLAY_COURSE_AND_SPEED is
  T_COURSE : FLOAT;
  INT_SPEED : INTEGER;
  T_SPEED : FLOAT;
begin
  COURSE_BUFFER.RECALL(T_COURSE);
  SPEED_BUFFER.RECALL(INT_SPEED);
  T_SPEED := FLOAT(INT_SPEED);
  CHECK_KEYBOARD.STOP;
  GOTOXY(DR, C1);
  PUT(FLOAT_TO_STRING(T_COURSE));
  GOTOXY(DR, C2);
  PUT(FLOAT_TO_STRING(T_SPEED));
  CHECK_KEYBOARD.CONTINUE;
end DISPLAY_COURSE_AND_SPEED;

procedure DISPLAY_BEARING_AND_DISTANCE is
  T_BEARING : FLOAT;
  T_DISTANCE : FLOAT;
  WP_NO : INTEGER;
begin
  CHECK_KEYBOARD.STOP;
  BEARING_BUFFER.RECALL(T_BEARING);
  DISTANCE_BUFFER.RECALL(T_DISTANCE);
  WP_NUMBER_BUFFER.RECALL(WP_NO);
  GOTOXY(DR, C1);
  PUT(FLOAT_TO_STRING(T_BEARING));
  GOTOXY(DR, C2 - 5);
  PUT(FLOAT_TO_STRING(T_DISTANCE));
  GOTOXY(DR, C2 + 20);
  PUT(INT_TO_CHAR(WP_NO));
  CHECK_KEYBOARD.CONTINUE;
end DISPLAY_BEARING_AND_DISTANCE;
```

```
  begin
    --display_handler
    loop
      accept MAKE_CHOICE(CHOICE : in CHARACTER) do
        NEW_CHOICE := CHOICE;
      end MAKE_CHOICE;
      if OLD_CHOICE /= NEW_CHOICE then
        case NEW_CHOICE is
          when '6' | '7' =>
            PREPARE_POSITION_DISPLAY;
          when '8' =>
            PREPARE_COURSE_SPEED_DISPLAY;
          when '9' =>
            PREPARE_BEARING_DISTANCE_DISPLAY;
          when others =>
            null;
        end case;
        OLD_CHOICE := NEW_CHOICE;
      end if;
      case NEW_CHOICE is
        when '1' =>
          ENTER_PRESENT_POSITION;
        when '2' =>
          ENTER_WAYPOINT;
        when '3' =>
          ENTER_COURSE;
        when '4' =>
          ENTER_SPEED;
        when '5' =>
          ENTER_STEER_TO_WAYPOINT;
        when '6' =>
          DISPLAY_PRESENT_POSITION;
        when '7' =>
          DISPLAY_WAYPOINT;
        when '8' =>
          DISPLAY_COURSE_AND_SPEED;
        when '9' =>
          DISPLAY_BEARING_AND_DISTANCE;
        when others =>
          null;
      end case;
    end loop;
  end DISPLAY_HANDLER;

begin  -- MAIN

-- initialize data buffers

  for WP_NO in WAYPOINT_RANGE loop
    WP_BUFFER(WP_NO).STORE(INITIAL_POSITION);
  end loop;
  COURSE_BUFFER.STORE(INITIAL_COURSE);
  SPEED_BUFFER.STORE(INITIAL_SPEED);
  BEARING_BUFFER.STORE(INITIAL_BEARING);
  DISTANCE_BUFFER.STORE(INITIAL_DISTANCE);
  WP_NUMBER_BUFFER.STORE(INITIAL_WP);
```

```
-- initialize screen and get initial user input

  PREPARE_SCREEN;
  GET_POSITION(0);
  GET_COURSE;
  GET_SPEED;
  SPECIAL_IO;

-- start tasks

  COMPUTE_POSITION.START;
  COMPUTE_BEARING_DISTANCE.START;
  CHECK_KEYBOARD.START;

end INS;
```

```
--------------------------------------------------------------------------------
-- UNIT_NAME              | mathutil.a
-- CSCI_NAME
-- UNIT_DESCRIPTION
--
--
-- UNIT_SPS_REFERENCE     | none
-- UNIT_CALLING_SEQUENCE
-- EXTERNAL_UNITS_CALLED  | none
-- INPUTS
-- OUTPUTS
-- CREATED                | 17 November 1988
-- AUTHOR                 | herbert guenterberg
-- DATE ------------ AUTHOR -------- REVISION # -- PR # -----TITLE ------------
--------------------------------------------------------------------------------
-- This package provides functions which are not specific to this application
-- and are not provided by the standard math library. The names of the
-- functions and procedures and their purpose are self explanatory. They are in
-- part taken from: Grady Booch; Software components with Ada.
--
--------------------------------------------------------------------------------

generic
  type NUMBER is digits <>;

  package FLOATING_POINT_UTILITIES is
    type BASE is range 2 .. 16;
    type NUMBERS is array (POSITIVE range <>) of NUMBER;

    function INTEGER_PART (THE_NUMBER : in NUMBER) return INTEGER;

    function REAL_PART (THE_NUMBER : in NUMBER) return NUMBER;

    function FLOOR (THE_NUMBER : in NUMBER) return INTEGER;

    function CEILING (THE_NUMBER : in NUMBER) return INTEGER;

    function IS_POSITIVE (THE_NUMBER : in NUMBER) return BOOLEAN;

    function IS_NEGATIVE (THE_NUMBER : in NUMBER) return BOOLEAN;

    function INT_TO_CHAR (INNUM : in INTEGER) return CHARACTER;

    function CHAR_TO_INT (INNUM : in CHARACTER) return INTEGER;

  end FLOATING_POINT_UTILITIES;


  package body FLOATING_POINT_UTILITIES is

    function INTEGER_PART (THE_NUMBER : in NUMBER) return INTEGER is
    begin
      if IS_NEGATIVE (THE_NUMBER) then
        return CEILING (THE_NUMBER);
      else
        return FLOOR (THE_NUMBER);
      end if;
    end INTEGER_PART;

    function REAL_PART (THE_NUMBER : in NUMBER) return NUMBER is
    begin
      return abs (THE_NUMBER - NUMBER(INTEGER_PART(THE_NUMBER)));
    end REAL_PART;
```

```
function FLOOR (THE_NUMBER : in NUMBER) return INTEGER is
  RESULT : INTEGER := INTEGER(THE_NUMBER);
begin
  if NUMBER(RESULT) > THE_NUMBER then
    return (RESULT - 1);
  else
    return RESULT;
  end if;
end FLOOR;

function CEILING (THE_NUMBER : in NUMBER) return INTEGER is
  RESULT : INTEGER := INTEGER(THE_NUMBER);
begin
  if NUMBER(RESULT) < THE_NUMBER then
    return (RESULT + 1);
  else
    return RESULT;
  end if;
end CEILING;

function IS_POSITIVE (THE_NUMBER : in NUMBER) return BOOLEAN is
begin
  return (THE_NUMBER > 0.0);
end IS_POSITIVE;

function IS_NEGATIVE (THE_NUMBER : in NUMBER) return BOOLEAN is
begin
  return (THE_NUMBER < 0.0);
end IS_NEGATIVE;

function INT_TO_CHAR (INNUM : in INTEGER) return CHARACTER is
  OUTNUM : CHARACTER := '0';
begin
  case INNUM is
    when 0 =>
      OUTNUM := '0';
    when 1 =>
      OUTNUM := '1';
    when 2 =>
      OUTNUM := '2';
    when 3 =>
      OUTNUM := '3';
    when 4 =>
      OUTNUM := '4';
    when 5 =>
      OUTNUM := '5';
    when 6 =>
      OUTNUM := '6';
    when 7 =>
      OUTNUM := '7';
    when 8 =>
      OUTNUM := '8';
    when 9 =>
      OUTNUM := '9';
    when others =>
      OUTNUM := '0';
  end case;
  return OUTNUM;
end INT_TO_CHAR;
```

```
  function CHAR_TO_INT (INNUM : in CHARACTER) return INTEGER is
    OUTNUM : INTEGER := 0;
  begin
    case INNUM is
      when '0' =>
        OUTNUM := 0;
      when '1' =>
        OUTNUM := 1;
      when '2' =>
        OUTNUM := 2;
      when '3' =>
        OUTNUM := 3;
      when '4' =>
        OUTNUM := 4;
      when '5' =>
        OUTNUM := 5;
      when '6' =>
        OUTNUM := 6;
      when '7' =>
        OUTNUM := 7;
      when '8' =>
        OUTNUM := 8;
      when '9' =>
        OUTNUM := 9;
      when others =>
        OUTNUM := 0;
    end case;
    return OUTNUM;
  end CHAR_TO_INT;
end FLOATING_POINT_UTILITIES;
```

```
--------------------------------------------------------------------------------
-- UNIT_NAME                | navutil.a
-- CSCI_NAME
-- UNIT_DESCRIPTION
--
--
-- UNIT_SPS_REFERENCE
-- UNIT_CALLING_SEQUENCE
-- EXTERNAL_UNITS_CALLED    | text_io, terminal, floating_point_utilities,
--                            data_storage
-- INPUTS
-- OUTPUTS
-- CREATED                  | 19 November 1988
-- AUTHOR                   | herbert guenterberg
-- DATE ------------ AUTHOR -------- REVISION # -- PR # -----TITLE ------------
--------------------------------------------------------------------------------
-- This package provides the routines needed in navigation programs in general.
--------------------------------------------------------------------------------

with DATA_STORAGE;

package NAVUTIL is
  MAX_WAYPOINTS : INTEGER := 3;
  subtype WAYPOINT_RANGE is INTEGER range 0 .. MAX_WAYPOINTS;
  type POSITION is
    record
      LATITUDE, LONGITUDE : FLOAT := 0.0;
    end record;
  subtype LAT_STR is STRING (1 .. 7);
  subtype LON_STR is STRING (1 .. 8);
  subtype SPEED_STR is STRING (1 .. 3);
  subtype COURSE_STR is STRING (1 .. 5);
  subtype OUT_STRING is STRING (1 .. 5);

  function FLOAT_TO_STRING(REAL_IN : in FLOAT) return OUT_STRING;

  procedure GET_POSITION (WP_NO : in WAYPOINT_RANGE);

  procedure GET_SPEED;

  procedure GET_COURSE;

  procedure DISPLAY_POSITION (T_POS : in POSITION);

  procedure BEARING_DISTANCE (POS1 : in POSITION; POS2 : in POSITION; BRG : out


    FLOAT; DIST : out FLOAT);

  procedure UPDATE_POSITION (INTERVAL : in DURATION; T_POS : in out POSITION;
    COURSE : in FLOAT; SPEED : in FLOAT);

  package POSITION_STORAGE is new DATA_STORAGE(POSITION);

  package FLOAT_STORAGE is new DATA_STORAGE(FLOAT);

  package INTEGER_STORAGE is new DATA_STORAGE(INTEGER);
```

```
    WP_BUFFER         : array (0 .. MAX_WAYPOINTS) of POSITION_STORAGE.BUFFER;
    COURSE_BUFFER     : FLOAT_STORAGE.BUFFER;
    SPEED_BUFFER      : INTEGER_STORAGE.BUFFER;
    BEARING_BUFFER    : FLOAT_STORAGE.BUFFER;
    DISTANCE_BUFFER   : FLOAT_STORAGE.BUFFER;
    WP_NUMBER_BUFFER  : INTEGER_STORAGE.BUFFER;

  end NAVUTIL;


  with TEXT_IO;
  use TEXT_IO;
  with TERMINAL;
  use TERMINAL;
  with MATH;
  use MATH;
  with FLOATING_POINT_UTILITIES;
  with DATA_STORAGE;

  package body NAVUTIL is

    package FLOATUTIL is new FLOATING_POINT_UTILITIES(FLOAT);
    use FLOATUTIL;

    function DEG_TO_RAD (DEG : in FLOAT) return FLOAT is
    begin
      return DEG * PI / 180.0;
    end DEG_TO_RAD;

    function RAD_TO_DEG (RAD : in FLOAT) return FLOAT is
    begin
      return RAD * 180.0 / PI;
    end RAD_TO_DEG;

    function FLOAT_TO_STRING(REAL_IN : in FLOAT) return OUT_STRING is
      INT      : INTEGER     := INTEGER_PART(REAL_IN);
      DECIMAL  : FLOAT       := REAL_PART(REAL_IN);
      T_STRING : OUT_STRING;
    begin
      T_STRING(1)  := INT_TO_CHAR(INT / 100);
      INT := INT mod 100;
      T_STRING(2)  := INT_TO_CHAR(INT / 10);
      T_STRING(3)  := INT_TO_CHAR(INT mod 10);
      T_STRING(4)  := '.';
      T_STRING(5)  := INT_TO_CHAR(INTEGER_PART(DECIMAL * 10.0));
      return T_STRING;
    end FLOAT_TO_STRING;
```

```
--------------------------------------------------------------------------
-- All procedures of name GET_*** receive an input string and convert it to
-- the appropriate data type
--------------------------------------------------------------------------

 procedure GET_POSITION (WP_NO : in WAYPOINT_RANGE) is
   T_LAT_S                : LAT_STR;
   T_LON_S                : LON_STR;
   LAT_DEG, LON_DEG       : INTEGER;
   LAT_MIN, LON_MIN       : FLOAT;
   SUCC1, SUCC2, SUCC3    : BOOLEAN  := FALSE;
   T_POS                  : POSITION := (0.0, 0.0);
 begin
   CLEAR_LINE(7, 5);
   GOTOXY(DR, C1);
   PUT("LATITUDE   N0000.0");
   GOTOXY(DR, C2);
   PUT("LONGITUDE W00000.0");
   while not SUCC3 loop
     T_LAT_S := "N0000.0";
     GOTOXY(DR, C1 + 11);
     PUT(T_LAT_S);
     GOTOXY(DR, C1 + 11);
     GET(T_LAT_S);
     if T_LAT_S(6) = '.' and (T_LAT_S(1) = 'N' or T_LAT_S(1) = 'n' or
      T_LAT_S(1) = 'S' or T_LAT_S(1) = 's') then
       SUCC1 := TRUE;
     else
       SUCC1 := FALSE;
     end if;
     LAT_DEG := CHAR_TO_INT(T_LAT_S(2)) * 10 + CHAR_TO_INT(T_LAT_S(3));
     LAT_MIN := (FLOAT(CHAR_TO_INT(T_LAT_S(4))) * 10.0 + FLOAT(CHAR_TO_INT(
       T_LAT_S(5))) * 1.0 + FLOAT(CHAR_TO_INT(T_LAT_S(7))) * 0.1) / 60.0;
     if LAT_MIN < 1.0 then
       SUCC2 := SUCC1 and TRUE;
     else
       SUCC2 := FALSE;
     end if;
     if (FLOAT(LAT_DEG) + LAT_MIN) <= 90.0 then
       SUCC3 := SUCC2 and TRUE;
     else
       SUCC3 := FALSE;
     end if;
     if T_LAT_S(1) = 'S' or T_LAT_S(1) = 's' then
       T_POS.LATITUDE := DEG_TO_RAD(FLOAT(LAT_DEG) + LAT_MIN) * (- 1.0);
     else
       T_POS.LATITUDE := DEG_TO_RAD(FLOAT(LAT_DEG) + LAT_MIN);
     end if;
   end loop;
   SUCC3 := FALSE;
   while not SUCC3 loop
     T_LON_S := "W00000.0";
     GOTOXY(DR, C2 + 10);
     PUT(T_LON_S);
     GOTOXY(DR, C2 + 10);
     GET(T_LON_S);
     if T_LON_S(7) = '.' and (T_LON_S(1) = 'W' or T_LON_S(1) = 'w' or
     T_LON_S (1) = 'E' or T_LON_S(1) = 'e') then
       SUCC1 := TRUE;
     else
       SUCC1 := FALSE;
     end if;
```

```
    LON_DEG := CHAR_TO_INT(T_LON_S(2)) * 100 + CHAR_TO_INT(T_LON_S(3)) * 10 +
      CHAR_TO_INT(T_LON_S(4));
    LON_MIN := (FLOAT(CHAR_TO_INT(T_LON_S(5))) * 10.0 + FLOAT(CHAR_TO_INT(
      T_LON_S(6))) * 1.0 + FLOAT(CHAR_TO_INT(T_LON_S(8))) * 0.1) / 60.0;
    if LON_MIN < 1.0 then
      SUCC2 := SUCC1 and TRUE;
    else
      SUCC2 := FALSE;
    end if;
    if (FLOAT(LON_DEG) + LON_MIN) <= 180.0 then
      SUCC3 := SUCC2 and TRUE;
    else
      SUCC3 := FALSE;
    end if;
    if T_LON_S(1) = 'E' or T_LON_S(1) = 'e' then
      T_POS.LONGITUDE := DEG_TO_RAD(FLOAT(LON_DEG) + LON_MIN) * (- 1.0);
    else
      T_POS.LONGITUDE := DEG_TO_RAD(FLOAT(LON_DEG) + LON_MIN);
    end if;
  end loop;
  WP_BUFFER(WP_NO).STORE(T_POS);
end GET_POSITION;

procedure GET_SPEED is
  SUCC    : BOOLEAN   := FALSE;
  SPEED_S : SPEED_STR := "000";
  SPEED_I : INTEGER   := 0;
begin
  CLEAR_LINE(DR, 3);
  GOTOXY(DR, C1);
  PUT("SPEED :");
  while not SUCC loop
    GOTOXY(DR, C1 + 8);
    PUT(SPEED_S);
    GOTOXY(DR, C1 + 8);
    GET(SPEED_S);
    SPEED_I := CHAR_TO_INT(SPEED_S(1)) * 100 + CHAR_TO_INT(SPEED_S(2)) * 10+
      CHAR_TO_INT(SPEED_S(3));
    if SPEED_I > 1 and SPEED_I < 500 then
      SUCC := TRUE;
    else
      SUCC := FALSE;
    end if;
  end loop;
  SPEED_BUFFER.STORE(SPEED_I);
end GET_SPEED;
```

```
procedure GET_COURSE is
    SUCC1, SUCC2 : BOOLEAN    := FALSE;
    COURSE_S     : COURSE_STR := "000.0";
    COURSE_F     : FLOAT      := 0.0;
  begin
    CLEAR_LINE(DR, 3);
    GOTOXY(DR, C1);
    PUT("COURSE :");
    while not SUCC2 loop
      GOTOXY(DR, C1 + 10);
      PUT(COURSE_S);
      GOTOXY(DR, C1 + 10);
      GET(COURSE_S);
      if COURSE_S(4) = '.' then
        SUCC1 := TRUE;
      else
        SUCC1 := FALSE;
      end if;
      COURSE_F := FLOAT(CHAR_TO_INT(COURSE_S(1)) * 100 +
CHAR_TO_INT(COURSE_S(2))      * 10 + CHAR_TO_INT(COURSE_S(3))) +
FLOAT(CHAR_TO_INT(COURSE_S(5))) * 0.1;       if COURSE_F >= 0.0 and COURSE_F <
359.9 then
        SUCC2 := SUCC1 and TRUE;
      else
        SUCC2 := FALSE;
      end if;
    end loop;
    COURSE_BUFFER.STORE(COURSE_F);
  end GET_COURSE;

-------------------------------------------------------------------------------
-- All procedures of name DISPLAY_*** take an input and convert it to a string
-- for screen output
-------------------------------------------------------------------------------

  procedure DISPLAY_POSITION (T_POS : in POSITION) is
    TEMPLAT                         : FLOAT := RAD_TO_DEG(T_POS.LATITUDE);
    TEMPLON                         : FLOAT := RAD_TO_DEG(T_POS.LONGITUDE);
    T_LAT_S                         : LAT_STR;
    T_LON_S                         : LON_STR;
    LAT_DEG, LON_DEG                : INTEGER;
    LAT_MIN, LON_MIN                : FLOAT;
    LAT_MIN_INT, LON_MIN_INT        : INTEGER;
    LAT_MIN_REAL, LON_MIN_REAL      : FLOAT;
  begin
    T_LAT_S(6) := '.';
    T_LON_S(7) := '.';
    if IS_NEGATIVE(TEMPLAT) then
      T_LAT_S(1) := 'S';
    else
      T_LAT_S(1) := 'N';
    end if;
    TEMPLAT := abs (TEMPLAT);
    LAT_DEG := INTEGER_PART(TEMPLAT);
    T_LAT_S(2) := INT_TO_CHAR(LAT_DEG / 10);
    T_LAT_S(3) := INT_TO_CHAR(LAT_DEG mod 10);
    LAT_MIN := REAL_PART(TEMPLAT) * 60.0;
    LAT_MIN_INT := INTEGER_PART(LAT_MIN);
    LAT_MIN_REAL := REAL_PART(LAT_MIN);
    T_LAT_S(4) := INT_TO_CHAR(LAT_MIN_INT / 10);
    T_LAT_S(5) := INT_TO_CHAR(LAT_MIN_INT mod 10);
    T_LAT_S(7) := INT_TO_CHAR(INTEGER_PART(LAT_MIN_REAL * 10.0));
    if IS_NEGATIVE(TEMPLON) then
```

```
      T_LON_S(1) := 'E';
    else
      T_LON_S(1) := 'W';
    end if;
    TEMPLON := abs (TEMPLON);
    LON_DEG := INTEGER_PART(TEMPLON);
    T_LON_S(2) := INT_TO_CHAR(LON_DEG / 100);
    LON_DEG := LON_DEG mod 100;
    T_LON_S(3) := INT_TO_CHAR(LON_DEG / 10);
    T_LON_S(4) := INT_TO_CHAR(LON_DEG mod 10);
    LON_MIN := REAL_PART(TEMPLON) * 60.0;
    LON_MIN_INT := INTEGER_PART(LON_MIN);
    LON_MIN_REAL := REAL_PART(LON_MIN);
    T_LON_S(5) := INT_TO_CHAR(LON_MIN_INT / 10);
    T_LON_S(6) := INT_TO_CHAR(LON_MIN_INT mod 10);
    T_LON_S(8) := INT_TO_CHAR(INTEGER_PART(LON_MIN_REAL * 10.0));
    GOTOXY(DR, C1);
    PUT(T_LAT_S);
    GOTOXY(DR, C2 - 5);
    PUT(T_LON_S);
  end DISPLAY_POSITION;

  procedure BEARING_DISTANCE (POS1 : in POSITION; POS2 : in POSITION; BRG : out
     FLOAT; DIST : out FLOAT) is
    LON1     : FLOAT := POS1.LONGITUDE;
    LON2     : FLOAT := POS2.LONGITUDE;
    LAT1     : FLOAT := POS1.LATITUDE;
    LAT2     : FLOAT := POS2.LATITUDE;
    LON_DIFF : FLOAT := LON2 - LON1;
    ARC_DIFF : FLOAT := 0.0;
    D        : FLOAT := 0.0;
    B        : FLOAT := 0.0;

 procedure DISTANCE (LAT1 : in FLOAT; LAT2 : in FLOAT; LON_DIFF : in FLOAT;
    DIST : out FLOAT) is
    D : FLOAT := 0.0;
    begin
      D := SIN(LAT1) * SIN(LAT2) + COS(LAT1) * COS(LAT2) * COS(LON_DIFF);
      if D /= 0.0 then
        D := - ARCTAN(SQRT(1.0 - D * D) / D) * 10800.0 / PI;
      end if;
      DIST := abs (D);
    end DISTANCE;
  begin
    DISTANCE(LAT1, LAT2, LON_DIFF, DIST);
    if LAT1 = LAT2 then
      if LON1 < LON2 then
        BRG := 270.0;
      else
        BRG := 90.0;
      end if;
    end if;
    if LON1 = LON2 then
      if LAT1 > LAT2 then
        BRG := 180.0;
      else
        BRG := 000.0;
      end if;
    else
      B := SIN(LON_DIFF) / (COS(LAT1) * SIN(LAT2) / COS(LAT2) - SIN(LAT1) *
        COS(LON_DIFF));
      B := ARCTAN(B) * 180.0 / PI;
    end if;
```

```
  if LON1 > LON2 and LAT1 > LAT2 then
    BRG := 180.0 - B;
  end if;
  if LON1 > LON2 and LAT1 < LAT2 then
    BRG := 0.0 - B;
  end if;
  if LON1 < LON2 and LAT1 > LAT2 then
    BRG := 180.0 - B;
  end if;
  if LON1 < LON2 and LAT1 < LAT2 then
    BRG := 360.0 - B;
  end if;
end BEARING_DISTANCE;

procedure UPDATE_POSITION (INTERVAL : in DURATION; T_POS : in out POSITION;
  COURSE : in FLOAT; SPEED : in FLOAT) is
  T_COURSE : FLOAT := DEG_TO_RAD((90.0 - COURSE));
  LAT_INC  : FLOAT := 0.0;
  LON_INC  : FLOAT := 0.0;
  DISTANCE : FLOAT := 0.0;
begin
  DISTANCE := SPEED / 3600.0 * FLOAT(INTERVAL);
  LAT_INC := DISTANCE * SIN(T_COURSE) / 60.0 * PI / 180.0;
  LON_INC := DISTANCE * COS(T_COURSE) / 60.0 * PI / 180.0;
  LON_INC := LON_INC / COS(T_POS.LATITUDE);
  if COURSE = 0.0 or COURSE = 360.0 or COURSE = 180.0 then
    T_POS.LATITUDE := T_POS.LATITUDE + LAT_INC;
  else
    if COURSE = 90.0 or COURSE = 270.0 then
      T_POS.LONGITUDE := T_POS.LONGITUDE - LON_INC;
    else
      T_POS.LATITUDE := T_POS.LATITUDE + LAT_INC;
      T_POS.LONGITUDE := T_POS.LONGITUDE - LON_INC;
    end if;
  end if;
end UPDATE_POSITION;

end NAVUTIL;
```

```
--------------------------------------------------------------------------------
-- UNIT_NAME              | TERM_S.A
-- CSCI_NAME
-- UNIT_DESCRIPTION       | SUPPORT TERMINAL INTERFACE
--
--
-- UNIT_SPS_REFERENCE
-- UNIT_CALLING_SEQUENCE
-- EXTERNAL_UNITS_CALLED  |
-- INPUTS
-- OUTPUTS
-- CREATED                | 17 November 1988
-- AUTHOR                 | herbert guenterberg, PUBLIC DOMAIN
-- DATE ------------ AUTHOR -------- REVISION # -- PR # -----TITLE ------------
--------------------------------------------------------------------------------
-- This package supplies the atomic functions and procedures used by the main
-- program to modify screen output to fit the application
--------------------------------------------------------------------------------

package TERMINAL is
  DASH_LINE : constant STRING :=
  "-----------------------------------------------------------------------------";


-- column and row definitions for screen output

  C1 : INTEGER := 5;
  C2 : INTEGER := 40;
  DR : INTEGER := 7;

-- UNIX specific procedures needed to allow monitoring keyboard interrupt

  procedure NORMAL_IO;

  procedure SPECIAL_IO;

-- clear the screen

  procedure CLEAR_SCREEN;

-- position the cursor anywhere on the screen

  procedure GOTOXY(ROW, COLUMN : in INTEGER);

-- takes the first line and the number of lines to be cleared

  procedure CLEAR_LINE(LINE, NUMBER : in INTEGER);

-- monitors keyboard interrupt has to be used in conjunction with NORMAL_IO
-- and SPECIAL_IO

  function KEY_PRESSED return BOOLEAN;

-- prepare the screen for different output modes

  procedure PREPARE_POSITION_DISPLAY;

  procedure PREPARE_COURSE_SPEED_DISPLAY;

  procedure PREPARE_BEARING_DISTANCE_DISPLAY;

  procedure PREPARE_SCREEN;
end TERMINAL;
```

```
-----------------------------------------------------------------------------
-- UNIT_NAME               | term_b.a
-- CSCI_NAME
-- UNIT_DESCRIPTION        | SUPPORT TERMINAL INTERFACE
--
--
-- UNIT_SPS_REFERENCE
-- UNIT_CALLING_SEQUENCE
-- EXTERNAL_UNITS_CALLED   | TEXT_IO, ASCII, CURSES, IOCTL, SYSTEM
-- INPUTS
-- OUTPUTS
-- CREATED                 | 17 November 1988
-- AUTHOR                  | herbert guenterberg / PUBLIC DOMAIN
-- DATE ------------ AUTHOR -------- REVISION # -- PR # -----TITLE ------------
-----------------------------------------------------------------------------
-- This package body is the only part of the program that contains TERMINAL
-- specific code
-----------------------------------------------------------------------------

with TEXT_IO;
use TEXT_IO;
with CURSES;
use CURSES;
with IOCTL;
use IOCTL;
with SYSTEM;
use SYSTEM;

package body TERMINAL is

  package INT_IO is new TEXT_IO.INTEGER_IO(INTEGER);
  use INT_IO;
  use ASCII;

  type TERMINAL_TYPE is (SUN, VT100);
  TERM : TERMINAL_TYPE := SUN;

  procedure NORMAL_IO is
  begin
    CURSES.ECHO;
    CURSES.NOCRMODE;
  end NORMAL_IO;

  procedure SPECIAL_IO is
  begin
    CURSES.NOECHO;
    CURSES.CRMODE;
  end SPECIAL_IO;

  procedure CLEAR_SCREEN is
  begin
    NEW_PAGE;
  end CLEAR_SCREEN;
```

```
procedure GOTOXY(ROW, COLUMN : in INTEGER) is
begin
  case TERM is
    when SUN =>
      PUT(ESC & "[");
      INT_IO.PUT(ROW, 1);
      PUT(';');
      INT_IO.PUT(COLUMN, 1);
      PUT('f');
    when VT100 =>
      PUT(ESC & "[");
      INT_IO.PUT(ROW, 1);
      PUT(';');
      INT_IO.PUT(COLUMN, 1);
      PUT('f');
  end case;
end GOTOXY;

procedure CLEAR_LINE(LINE, NUMBER : in INTEGER) is
begin
  GOTOXY(LINE, 1);
  for I in 1 .. NUMBER loop
    for J in 1 .. 79 loop
      TEXT_IO.PUT(" ");
    end loop;
    NEW_LINE;
  end loop;
end CLEAR_LINE;

function KEY_PRESSED return BOOLEAN is
  GO : INTEGER;
  INT_VAR : INTEGER := 0;
  A : SYSTEM.ADDRESS := INT_VAR'ADDRESS;
begin
  GO := IOCTL.IOCTL(0, FIONREAD, A);
  return INT_VAR > 0;
end KEY_PRESSED;

procedure PREPARE_POSITION_DISPLAY is
begin
  CLEAR_LINE(DR, 3);
  GOTOXY(8, C1);
  TEXT_IO.PUT("-------");
  GOTOXY(8, C2 - 5);
  TEXT_IO.PUT("-------");
  GOTOXY(8, C2 + 20);
  TEXT_IO.PUT("----------------");
  GOTOXY(9, C1);
  TEXT_IO.PUT(" LAT");
  GOTOXY(9, C2 - 5);
  TEXT_IO.PUT(" LONG");
  GOTOXY(9, C2 + 20);
  TEXT_IO.PUT(" POSITION");
end PREPARE_POSITION_DISPLAY;
```

```
procedure PREPARE_COURSE_SPEED_DISPLAY is
begin
  CLEAR_LINE(DR, 3);
  GOTOXY(8, C1);
  TEXT_IO.PUT("-------");
  GOTOXY(8, C2);
  TEXT_IO.PUT("-------");
  GOTOXY(9, C1);
  TEXT_IO.PUT(" COURSE");
  GOTOXY(9, C2);
  TEXT_IO.PUT(" SPEED");
end PREPARE_COURSE_SPEED_DISPLAY;

procedure PREPARE_BEARING_DISTANCE_DISPLAY is
begin
  CLEAR_LINE(DR, 3);
  GOTOXY(8, C1);
  TEXT_IO.PUT("-------");
  GOTOXY(8, C2 - 5);
  TEXT_IO.PUT("-------");
  GOTOXY(8, C2 + 20);
  TEXT_IO.PUT("-------");
  GOTOXY(9, C1);
  TEXT_IO.PUT(" BRG");
  GOTOXY(9, C2 - 5);
  TEXT_IO.PUT(" DIST");
  GOTOXY(9, C2 + 20);
  TEXT_IO.PUT(" TO WP");
end PREPARE_BEARING_DISTANCE_DISPLAY;

procedure PREPARE_SCREEN is
begin
  INITSCR;
  CLEAR_SCREEN;
  GOTOXY(1, 27);
  TEXT_IO.PUT("I N S   S I M U L A T O R");
  GOTOXY(2, 1);
  TEXT_IO.PUT(DASH_LINE);
  GOTOXY(14, 1);
  TEXT_IO.PUT(DASH_LINE);
  GOTOXY(16, C1);
  TEXT_IO.PUT("ENTER / UPDATE");
  GOTOXY(16, C2);
  TEXT_IO.PUT("DISPLAY");
  GOTOXY(17, C1);
  TEXT_IO.PUT("--------------");
  GOTOXY(17, C2);
  TEXT_IO.PUT("--------------");
  GOTOXY(19, C1);
  TEXT_IO.PUT("[1] PRESENT POSITION");
  GOTOXY(20, C1);
  TEXT_IO.PUT("[2] WAYPOINT");
  GOTOXY(21, C1);
  TEXT_IO.PUT("[3] COURSE");
  GOTOXY(22, C1);
  TEXT_IO.PUT("[4] SPEED");
  GOTOXY(23, C1);
  TEXT_IO.PUT("[5] STEER TO WAYPOINT");
  GOTOXY(19, C2);
  TEXT_IO.PUT("[6] PRESENT POSITION");
  GOTOXY(20, C2);
  TEXT_IO.PUT("[7] WAYPOINT");
  GOTOXY(21, C2);
```

```
    TEXT_IO.PUT("[8] COU-SE / SPEED");
    GOTOXY(22, C2);
    TEXT_IO.PUT("[9] BEARING / DISTANCE");
  end PREPARE_SCREEN;
end TERMINAL;
```

```
---------------------------------------------------------------------------------
-- UNIT_NAME              | data_sto.ada
-- CSCI_NAME
-- UNIT_DESCRIPTION         data structure to store data in tasks
--
--
-- UNIT_SPS_REFERENCE
-- UNIT_CALLING_SEQUENCE
-- EXTERNAL_UNITS_CALLED
-- INPUTS
-- OUTPUTS
-- CREATED                | 15 January 1989
-- AUTHOR                 | herbert guenterberg
-- DATE ------------ AUTHOR -------- REVISION # -- PR # -----TITLE ------------
---------------------------------------------------------------------------------
-- This package supplies the necessary data structure to store data in a way,
-- that allows more than one task to access these data, without the risk of
-- accessing invalid data, or more than one task trying to modify the same
-- data at the same time. The implementation is generic to allow for different
-- data types to be stored.
-- The algorithm was taken from: David A.Watt and others; Ada Language and
-- Methodologie; Prentice Hall; 1987
---------------------------------------------------------------------------------

generic
  type ITEM_TYPE is private;

  package DATA_STORAGE is

    task type BUFFER is
      entry STORE(ITEM  : in ITEM_TYPE);
      entry RECALL(ITEM : out ITEM_TYPE);
    end BUFFER;
  end DATA_STORAGE;

  package body DATA_STORAGE is

    task body BUFFER is
      DATUM : ITEM_TYPE;
    begin
      loop
        select
          accept STORE (ITEM : in ITEM_TYPE) do
            DATUM := ITEM;
          end STORE;
        or
          accept RECALL (ITEM : out ITEM_TYPE) do
            ITEM := DATUM;
          end RECALL;
        end select;
      end loop;
    end BUFFER;
  end DATA_STORAGE;
```

LIST OF REFERENCES

[1] Grady Booch, *Software Engineering with ADA*, The Benjamin/Cummings Publishing Company, 1986.

[2] M.T.Devlin, *Introducing ADA: Problem and Potentials*, USAF Satellite Control Facility, unpublished report, 1980.

[3] Hank Raum, *Design and Implementation of an Expert User Interface for CAPS*, MS Thesis, Naval Postgraduate School, Monterey, CA, December 1988.

[4] Dan Galik, *A Conceptual Design of a Software Base Management System for CAPS* , MS Thesis, Naval Postgraduate School, Monterey, CA, December 1988.

[5] Laura Marlowe, *A Scheduler for Critical Time Constraints*, MS Thesis, Naval Postgraduate School, Monterey, CA, December 1988.

[6] Charles Altizer, *Implementation of a Language Translator for CAPS*, MS Thesis, Naval Postgraduate School, Monterey, CA, December 1988.

[7] Mary Lou Wood, *Runtime Support for Rapid Prototyping*, MS Thesis, Naval Postgraduate School, Monterey, CA, December 1988.

[8] Roger Thorstensen, *A Graphical Editor for CAPS*, MS Thesis, Naval Postgraduate School, Monterey, CA, December 1988.

[9] Valdis Berzins,*"Software Engineering"*, class notes provided at Naval Postgraduate School, Monterey, CA, Spring Quarter 1988.

[10] *ADA Language Reference Manual ANSI/MIL-STD-1815A.*

[11] David A. Watt, Brian A. Wichman and William Findlay, *ADA Language and Methodology*, Prentice Hall, 1987.

[12] Grady Booch, *Software Components with ADA*, The Benjamin/Cummings Publishing Company, 1987.

[13] V. Berzins, Luqi *Semantics of a Real-Time Language*, in Proceedings of IEEE 9th Real-Time Symposium, Refereed Paper, Huntsville, AL, December 6-8, 1988.

[14] Luqi, V.Berzins, *Rapidly Prototyping Real Time Systems*, IEEE Software, pp 25-36, September 1988.

[15] Luqi, *Handling Timing Constraints in Rapid Prototyping*, in proceedings of 22nd Annual Hawaii International Conference on System Sciences, Refereed Paper, Kailua-Kona, Hawaii, Januar 1989.

[16] Luqi, V. Berzins, *Execution of a High Level Real-Time Language*, in Proceedings of IEEE 9th Real-Time Symposium, Refereed Paper, Huntsville, AL, December 6-8, 1988.

[17] Luqi, V. Berzins, *Execution of a High Level Real-Time Language*, in Proceedings of IEEE 9th Real-Time Symposium, Refereed Paper, Huntsville, AL, December 6-8, 1988.

[18] Jack Schwartz, Presentation at the CPS/CPL Program Briefing, Vienna, Virginia, 21 February 1989 by DARPA/ISTO

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Office of Naval Research Office of the Chief of Naval Research Code 1224 800 N. Quincy Street Arlington, Virginia 22217-5000	1
4.	Ada Joint Program Office OUSDRE(R&AT) Pentagon Washington, D.C. 20301	1
5.	Naval Sea Systems Command CAPT Thompson National Center #2, Suite 7N06 Washington, D.C. 22202	1
6.	Office of the Secretary of Defense COL Green STARS Program Office Washington, D.C. 20301	1
7.	Office of the Secretary of Defense R&AT/S&CT, RM 3E114 STARS Program Office Washington, D.C. 20301	1
8.	Commanding Officer Naval Research Laboratory Code 5150 Attn. Dr. Elizabeth Wald Washington, D.C. 20375-5000	1
9.	Navy Ocean System Center Attn. Linwood Sutton, Code 423 San Diego, California 92152-500	1

10.	Navy Ocean System Center Attn. Les Anderson, Code 413 San Diego, California 92152-500	1
11.	National Science Foundation Attn. Dr. William Wulf Washington, D.C. 20550	1
12.	National Science Foundation Division of Computer and Computation Research Attn. Tom Keenan Washington, D.C. 20550	1
13.	Office of Naval Research Computer Science Division, Code 1133 Attn. Dr. Van Tilborg 800 N. Quincy Street Arlington, Virginia 22217-5000	1
14.	Office of Naval Research Computer Science Division, Code 1133 Attn. Dr. R.Wachten 800 N. Quincy Street Arlington, Virginia 22217-5000	1
15.	Office of Naval Research Applied Mathematics and Computer Science, Code 1211 Attn. Dr J. Smith 800 N. Quincy Street Arlington, Virginia 22217-5000	1
16.	New Jersey Institute of Technology Computer Science Department Attn. Dr. Peter Ng Newark, New Jersey 07102	1
17.	Southern Methodist University Computer Science Department Attn. Dr. Murat Tanik Dallas, Texas 75275	1
18.	Editor-in-Chief, IEEE Software Attn. Dr. Ted Lewis Oregon State University Computer Science Department Corvallis, Oregon 97331	1
19.	University of Texas at Austin Computer Science Department Attn. Dr. Al Mok Austin, Texas 78712	1

20. University of Maryland 1
College of Business Management
Tydings Hall, Room 0137
Attn. Dr. Alan Hevner
College Park, Maryland 20742

21. University of California at Berkeley 1
Department of Electrical Engineering and Computer Science
Computer Science Division
Attn. Dr. C.V. Ramamoorthy
Berkeley, California 94720

22. University of California at Los Angeles 1
School of Engineering and Applied Science
Computer Science Department
Attn. Dr. Daniel Berry
Los Angeles, California 90024

23. University of Maryland 1
Computer Science Department
Attn. Dr. Y. H. Chu
College Park, Maryland 20742

24. University of Maryland 1
Computer Science Department
Attn. Dr. N. Roussapoulos
College Park, Maryland 20742

25. Kestrel Institute 1
Attn. Dr. C. Green
1801 Page Mill Road
Palo Alto, California 94304

26. Massachusetts Institute of Technology 1
Department of Electrical Engineering and Computer Science
545 Tech Square
Attn. Dr. B. Liskov
Cambridge, Massachusetts 02139

27. Massachusetts Institute of Technology 1
Department of Electrical Engineering and Computer Science
545 Tech Square
Attn. Dr. J. Guttag
Cambridge, Massachusetts 02139

28. University of Minnesota 1
Computer Science Department
136 Lind Hall
207 Church Street SE
Attn. Dr. Slagle
Minneapolis, Minnesota 55455

29. International Software Systems Inc. 1
12710 Research Boulevard, Suite 301
Attn. Dr. R. T. Yeh
Austin, Texas 78759

30. Software Group, MCC 1
9430 Research Boulevard
Attn. Dr. L. Belady
Austin, Texas 78759

31. Carnegie Mellon University 1
Software Engineering Institute
Department of Computer Science
Attn. Dr. Lui Sha
Pittsburgh, Pennsylvania 15260

32. IBM T. J. Watson Research Center 1
Attn. Dr. A. Stoyenko
P.O. Box 704
Yorktown Heights, New York 10598

33. The Ohio State University 1
Department of Computer and Information Science
Attn. Dr. Ming Liu
2036 Neil Ave Mall
Columbus, Ohio 43210-1277

34. University of Illinois 1
Department of Computer Science
Attn. Dr. Jane W. S. Liu
Urbana Champaign, Illinois 61801

35. University of Massachusetts 1
Department of Computer and Information Science
Attn. Dr. John A. Stankovic
Amherst, Massachusetts 01003

36. University of Pittsburgh 1
Department of Computer Science
Attn. Dr. Alfs Berztiss
Pittsburgh, Pennsylvania 15260

37. Defense Advanced Research Projects Agency (DARPA) 1
Integrated Strategic Technology Office (ISTO)
Attn. Dr. Jacob Schwartz
1400 Wilson Boulevard
Arlington, Virginia 22209-2308

38. Defense Advanced Research Projects Agency (DARPA) 1
Integrated Strategic Technology Office (ISTO)
Attn. Dr. Squires
1400 Wilson Boulevard
Arlington, Virginia 22209-2308

39. Defense Advanced Research Projects Agency (DAF.PA) 1
Director, Naval Technology Office
1400 Wilson Boulevard
Arlington, Virginia 2209-2308

42. Defense Advanced Research Projects Agency (DARPA) 1
Director, Prototype Projects Office
1400 Wilson Boulevard
Arlington, Virginia 2209-2308

43. Defense Advanced Research Projects Agency (DARPA) 1
Director, Tactical Technology Office
1400 Wilson Boulevard
Arlington, Virginia 2209-2308

44. MCC AI Laboratory 1
Attn. Dr. Michael Gray
3500 West Balcones Center Drive
Austin, Texas 78759

45. COL C. Cox, USAF 1
JCS (J-8)
Nuclear Force Analysis Division
Pentagon
Washington, D.C. 20318-8000

47. University of California at San Diego 1
Department of Computer Science
Attn. Dr. William Howden
La Jolla, California 92093

48. University of California at Irvine 1
Department of Computer and Information Science
Attn. Dr. Nancy Levenson
Irvine, California 92717

49. University of California at Irvine 1
Department of Computer and Information Science
Attn. Dr. L. Osterweil
Irvine, California 92717

50. University of Colorado at Boulder 1
Department of Computer Science
Attn. Dr. Lloyd Fosdick
Boulder, Colorado 80309-0430

51. Santa Clara University 1
Department of Electrical Engineering and Computer Science
Attn. Dr. M. Ketabchi
Santa Clara, California 95053

52. Oregon Graduate Center 1
Portland (Beaverton)
Attn. Dr. R. Kieburtz
Portland, Oregon 97005

54. Dr. Bernd Kraemer 1
GMD
Postfach 1240
D-5205 Schloss Birlinghaven
Sankt Augustin 1, West Germany

55. Dr. Aimram Yuhudai 1
Tel Aviv University
School of Mathematical Sciences
Department of Computer Science
Tel Aviv, Israel 69978

56. Dr. Robert M. Balzer 1
USC-Information Sciences Institute
4676 Admiralty Way
Suite 1001
Marina del Ray, California 90292-6695

57. U.S. Air Force Systems Command 1
Rome Air Development Center
Attn. Frank Lamonica
Griffis Air Force Base, New York 13441-5700

58. U.S. Air Force Systems Command 1
Rome Air Development Center
RADC/COE
Attn. Mr. William E. Rzepka
Griffis Air Force Base, New York 13441-5700

59. Commanding Officer 1
GENAVAIRWING 3
Feuerweg 6
2859 Nordholz, West-Germany

60. LCDR H. Günterberg 1
GENAVAIRWING 3
-Stab Fliegende Gruppe-
Feuerweg 6
2859 Nordholz, West-Germany

61. Commanding Officer
KdoMFüSys
Wibbelhofstr. 3
2940 Wilhelmshaven, West-Germany 1